

# Effective Density Queries for Moving Objects in Road Networks<sup>\*</sup>

Caifeng Lai<sup>1,2</sup>, Ling Wang<sup>1,2</sup>, Jidong Chen<sup>1,2</sup>, Xiaofeng Meng<sup>1,2</sup>, and  
Karine Zeitouni<sup>3</sup>

<sup>1</sup> School of Information, Renmin University of China

<sup>2</sup> Key Laboratory of Data Engineering and Knowledge Engineering, MOE  
{laicf, jingyiwang, chenjd, xfmeng}@ruc.edu.cn

<sup>3</sup> PRISM, Versailles Saint-Quentin University, France  
karine.zeitouni@prism.uvsq.fr

**Abstract.** Recent research has focused on density queries for moving objects in highly dynamic scenarios. An area is dense if the number of moving objects it contains is above some threshold. Monitoring dense areas has applications in traffic control systems, bandwidth management, collision probability evaluation, etc. All existing methods, however, assume the objects moving in the Euclidean space. In this paper, we study the density queries in road networks, where density computation is determined by the length of the road segment and the number of objects on it. We define an effective road-network density query guaranteeing to obtain useful answers. We then propose the cluster-based algorithm for the efficient computation of density queries for objects moving in road networks. Extensive experimental results show that our methods achieve high efficiency and accuracy for finding the dense areas in road networks.

## 1 Introduction

The advances in mobile communication and database technology have enabled innovative mobile applications monitoring moving objects. In some applications, the object movement is constrained by an underlying spatial network, e.g., vehicles move on road networks and trains on railway networks. In this scenario, objects can not move freely in space, and their positions must satisfy the network constrains. A network is usually modeled by a graph representation, comprising a set of nodes (intersections) and a set of edges (segments). Depending on the application, the graph may be directed, i.e. each edge has an orientation. Additionally, moving objects are assumed to move in a piecewise linear manner [6], i.e., each object moves at a stable velocity at each edge. The distance between two arbitrary objects is defined as the network distance between them on the network. Several types of queries have been studied in the road network, such as kNN queries [7], range queries [9], aggregate nearest neighbor queries [11], reverse nearest neighbor queries [12].

In this paper, we focus on the problem of dynamic density queries for moving objects in road networks. The objective is to find dense areas with high concentration of moving objects in a road network efficiently. The density query can

---

<sup>\*</sup> This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091, 60273018; Program for New Century Excellent Talents in University (NCET).

be used in the traffic management systems to identify and predict the congested areas or traffic jams. For example, the transportation bureau needs to monitor the dense regions periodically in order to discover the traffic jams.

Existing research works on the density query [2, 4] assume the objects moving in a free style and define the density query in the Euclidean space. In this setting, the general density-based queries are difficult to be answered efficiently and their focus is hence turned to simplified queries [2] or specialized density queries without answer loss [4]. These methods use the grid to partition the data space into disjoint cells and report the rectangle area with the fixed size. However, the real dense areas may be larger or smaller than the fixed-size rectangle and appear in different shapes. Simplifying the dense query to return the area with fixed size and shape can not reflect the natural congested area in real-life application. We focus on the density query in the road-network setting, where the dense area consists of road segments containing large number of moving objects and may be formed in any size and shape. The real congested areas can therefore be obtained by finding the dense segments. In addition, for querying objects moving in a road network, the existing methods based on a regular spatio-temporal grid ignore the network constraint and therefore result in inaccurate query results.

Considering the real-life application, finding dense regions for a point in time is more useful than finding the dense regions for a period of time [4]. In this paper, we study the querying for dense regions consisting of dense segments for a point in time. For monitoring the dense areas of moving objects in the road network, the dense query requests need to be issued periodically in order to find the changes of dense areas. If we use the existing methods, the total cost is quite high since each query request requires accessing all objects in the road network. Since clustering can represent the dense areas naturally, we propose a cluster-based method to process density queries in a road network. The moving objects are first grouped into cluster units on each road segment according to their locations and moving patterns. Then the cluster units are maintained continuously. The process can be treated as a separate pre-processing for the periodical density queries. For density query processing, we use a two-phase algorithm to identify the dense areas based on the summary information of the cluster units. Maintaining cluster units comes with a cost, but our experimental evaluations demonstrate it is much cheaper than keeping the complete information about individual locations of objects to process the dynamic density queries.

Our contributions are summarized as follows:

- We define the density query for moving objects in road networks that is amenable to obtain the effective answers.
- We propose a cluster-based algorithm to efficiently monitor the dense areas in a road network.
- We show, through extensive experiments, that our query algorithms achieve high efficiency and accuracy.

The rest of the paper is organized as follows. Section 2 reviews related work on density query processing and clustering moving objects. Section 3 gives the problem definition. Section 4 details our density query method including dynamic cluster maintenance and two-phase query algorithm. Algorithm analysis and experimental results are shown in Section 5. We conclude this paper in Section 6.

## 2 Related Work

Density query for moving objects is first proposed in [2]. The objective is to find regions in space and time that with the density higher than a given threshold. They find the general density-based queries difficult to be answered efficiently and hence turn to simplified queries. Specifically, they partition the data space into disjoint cells, and the simplified density query reports cells, instead of arbitrary regions that satisfy the query conditions. This scheme may result in answer loss. To solve this problem, Jensen et al. [4] define an effective density query to guarantee that there is no answer loss. The two works both assume the objects moving in a free style and define the density query in Euclidean space. However, efficient dynamic density query in spatial networks is crucial for many applications. As an example of a real world, considering that the queries correspond to vehicles distribution in the road network, the users would like to know real-time traffic density distribution. Clearly, in this case the Euclidean density query methods are inapplicable, since the path between two cars is restricted by the underlying road network. Additionally, these existing query methods can not reflect the natural dense area in a road network since they simplify the dense query to return the area with fixed size and shape. The grid-based algorithms also ignore the network constraint and result in inaccurate query results. It is natural to represent the dense area in a road network as road segments containing large number of moving objects. We exploit the network property and define *effective road-network density query* (*e-RNDQ*) to return the natural density areas with arbitrary size and shape in the road network.

Existing network based clustering algorithms are also related to our work. Jain et al. [3] use the agglomerative hierarchical approach to cluster nodes of a graph. CHAMELEON [5] is a general-purpose algorithm, which transforms the problem space into a weighted *kNN* graph, where each object is connected with its *k* nearest neighbors. Yiu and Mamoulis [10] define the problem of clustering objects based on the network distance and propose algorithms for three different clustering paradigms, i.e., *k-medoids* for K-partitioning,  *$\epsilon$ -link* for density-based, and *single-link* for hierarchical clustering. The  *$\epsilon$ -link* method is most efficient to find dense segments in road network. However, all these solutions assumed a static dataset. Li *et al.* [6] propose the micro moving cluster (MMC) to clustering moving objects in Euclidean spaces. Our clustering algorithm focuses on moving objects in the road network which exploits the road-network features and provides the summary information of moving objects to density query processing.

There are some other related works on query processing in spatial network databases [1, 7, 11]. Their focus is to evaluate various types of queries based on the network distance by minimizing the cost of the expensive shortest path computation. To the best of our knowledge, this is the first work which specifies on the cluster-based method for dynamic density queries in spatial networks.

## 3 Problem Definition

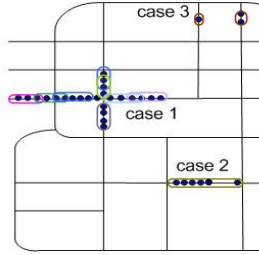
As the result of density queries in the road network are the set of dense segments, we first introduce the concepts of *density* and *dense segment*.

**Definition 1** *Density.* The density of a road segment *s* is represented as  $density(s) = N/len(s)$ , where *N* is the number of objects on *s* and  $len(s)$  is the length of *s*.

**Definition 2** *Dense Segment (DS)*. The road segment  $s$  is a dense segment (DS) if and only if  $\text{density}(s) \geq \rho$ , where  $\rho$  is a user-defined density parameter.

A straightforward method to process the query is to traverse all objects moving on a road network to compute dense regions by their number, the length of the segment and a given density threshold. Figure 1 shows a density query in a road network. Obviously, the cost is very high and it is difficult to find effective results. Specifically, the query results may have three problems as follows:

- 1) The different DS may be overlapped, such as Case 1 in Figure 1.
- 2) The distribution of moving objects may be very skewed in some DS, namely, the distribution of objects is dense in one part of a DS, but it is sparse in another part, such as Case 2 in Figure 1.
- 3) Some DS may contain very few objects, such as Case 3 in Figure 1.



**Fig. 1.** An example of density query

Such query results are less useful. Thus, we define an *effective density query* in a road network to find the useful dense regions with a high concentration of objects and symmetrical distribution of objects as well as no overlapping.

**Definition 3** *Effective Road-Network Density Query (e-RNDQ)*: Given density parameter  $\rho$ , find all dense segments that satisfy the following conditions:

1. Any dense segment set can not be intersecting (namely no overlaps).
2. In each dense segment set, the distance between any neighboring object is not more than a given distance threshold  $\delta$ .
3. The length of dense segments is not less than a given length threshold  $L$ .
4. Any dense segment containing moving objects is in the query result set.

The first condition ensures that the result is not redundant. It avoids the case 1 in Figure 1. The second condition guarantees that objects are symmetrically distributed in a dense segment set. The third condition provides restriction that there is no small segments that only contain few objects in the result. The fourth condition ensures that query results do not suffer from answer loss.

## 4 Density Query Processing in Road Networks

### 4.1 Overview

Considering the feature of road networks, we propose a cluster-based density querying algorithm, which regards clustering operation as a pre-processing to provide the summary information of moving objects. In the query processing, we develop a two-phase filter-and-refinement algorithm to find dense areas.

## 4.2 Cluster-based Query Preprocessing

To reduce the cost of clustering maintenance, we introduce the definition of Cluster Unit. A cluster unit is a group of moving objects close to each other at present and near future time. It will be incrementally maintained with moving of objects in it. Specifically, we constrain the objects in a cluster unit moving in the same direction and on the same segment. For keeping the objects in a cluster unit dense enough, the network distance between each pair of neighboring objects in a cluster unit does not exceed a system threshold  $\epsilon$ . As mentioned in Introduction, we assume that objects move in a piecewise linear manner and the next segment to move along is known in advance. Formally, a cluster unit is defined as follows:

**Definition 4** *Cluster Unit (CU).* A cluster unit is represented by  $CU = (O, n_a, n_b, head, tail, ObjNum)$ , where  $O$  is a list of objects  $\{o_1, o_2, \dots, o_i, \dots, o_n\}$ ,  $o_i = (oid_i, n_a, n_b, pos_i, speed_i, next\_node_i)$ , where  $pos_i$  is the relative location to  $n_a$ ,  $speed_i$  is the moving speed and  $(n_b, next\_node)$  is the next segment to move along. Without loss of generality, assuming  $pos_1 \leq pos_2 \leq \dots \leq pos_n$ , it must satisfy  $|pos_{i+1} - pos_i| \leq \epsilon$  ( $1 \leq i \leq n - 1$ ). Since all objects are on the same segment  $(n_a, n_b)$ , the position of the CU is determined by an interval  $(head, tail)$  in terms of the network distance from  $n_a$ . Thus, the length of the CU is  $|tail - head|$ .  $ObjNum$  is the number of objects in the CU.

Initially, based on the definition, a set of CU are created by traversing all segments in the network and their associated objects. The CUs are incrementally maintained after their creation. As time elapsed, the distance between adjacent objects in a CU may exceed  $\epsilon$ . Thus, we need to split the CU. A CU may also merge with its adjacent CUs when they are within the distance of  $\epsilon$ . Hence, for each CU, we predict the time when they may split or merge. The predicted split and merge events are then inserted into an event queue. Afterwards, when the first event in the queue takes place, we process it and update the affected CUs. This process is continuously repeated. The key problems are: 1) how to predict split/merge time of a CU, and 2) how to process a split/merge event of a CU.

The split of a CU may occur in two cases. The first one is when CU arriving at the end of the segment (i.e., an intersection node of the road network). When the moving objects in a CU reach an intersection node, the CU has to be split since they may head in different directions. Obviously, a split time is the time when the first object in the CU arrives at the node. In the second case, the split of a CU is when the distance between some neighboring objects moving on the segment exceed  $\epsilon$ . However, it is not easy to predict the split time since the neighborhood of objects changes over time. Therefore, the main task is to dynamically maintain the order of objects on the segment. We compute the earliest time instance when two adjacent objects in the CU meet as  $t_m$ . We then compare the maximum distance between each pair of adjacent objects with  $\epsilon$  until  $t_m$ . if this distance exceeds  $\epsilon$  at some time, the process stops and the earliest time exceeding  $\epsilon$  is recorded as the split time of CU. Otherwise, we update the order of objects starting from  $t_m$  and repeat the same process until some distance exceed  $\epsilon$  or one of the objects arrives at the end of the segment. When the velocity of an object changes over the segment, we need to re-predict the split and merge time of the CU.

To reduce the processing cost of splitting at the end of segment, we propose group split scheme. When the first object leaves the segment, we split the original

CU into several new CUs according to objects' directions (which can be implied by *next\_node*). On one hand, we compute a to-be-expired time (i.e., the time until the departure from the segment) for each object in the original CU and retain the CU until the last object leaves the segment. On the other hand, we attach a to-be-valid time (with the same value as *to-be-expired time*) for each object in the new CUs. Only valid objects will be counted in constructing CUs.

The merge of CUs may occur when adjacent CUs in a segment are moving together (i.e., their network distance  $\leq \epsilon$ ). To predict the initial merge time of CUs, we dynamically maintain the boundary objects of each CU and their validity time (the period when they are treated as boundary of the CU), and compare the minimum distance between the boundary objects of two CUs with the threshold  $\epsilon$  at their validity time. The boundary objects of CUs can be obtained by maintaining the order of objects during computing the split time.

The processing of the merge event is similar to the split event on the segment. We get the merge event and time from the event queue to merge the CUs into one CU and compute the split time and merge time of the merged CU. Finally, the corresponding affected CUs in the event queue are updated.

Besides the split and merge of CUs, new objects may come into the network or existing objects may leave. For a new object, we locate all CUs of the same segment that the object enters and see if the new object can join any CU according to the CU definition. If the object can join some CU, its split and merge events are updated. If no such CUs are found, a new CU for the object is created and the merge event is computed. For a leaving object, we update the split and merge events of its original CU if necessary. Due to the limitation of the space, we omit the algorithm pseudo of maintaining CUs.

### 4.3 Density Query Processing

Based on the dynamic CUs, density query at any time point can be processed efficiently to return dense areas in the road networks. And then the dense segment (DS) we defined in Section 3 is represented as  $(CU, n_a, n_b, startpos, endpos, len, N)$ , where  $CU$  is the set of cluster units on segment  $(n_a, n_b)$ ,  $startpos$  is the start position of the DS, and  $endpos$  is the end position of the DS,  $len$  is the length of DS,  $N$  is the number of objects. To obtain the effective dense areas restricted in the e-RNDQ, we introduce the parameter  $\delta$  to DS.

**Definition 5**  *$\delta$ -Dense Segment ( $\delta$ -DS).* A DS is  $\delta$ -DS if and only if the distance between any adjacent CUs is not more than  $\delta$  (i.e. guarantee that the distance between any adjacent object satisfies  $Distance(o_i, o_{i+1}) \leq \delta$ ), and density is not less than  $\rho$ . (For convenience, we abbreviate the term  $\delta$ -DS to DS in the sequel)

In fact,  $\delta$  is a user-defined parameter of the density query and  $\epsilon$  is a system parameter to maintain the CUs. Since the distance of adjacent objects is not more than  $\epsilon$  in a CU, in order to retrieve dense areas based on CUs, we require  $\epsilon \leq \max\{\delta, \frac{1}{\rho}\}$ . In the road network, a dense area is represented as a dense segment set, which may contain several DSs in different segments. Therefore, we exploit network nodes to optimize the combination of these DSs.

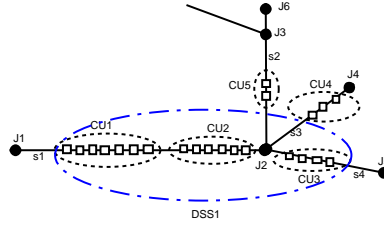
**Definition 6**  *$\delta$ -ClusterNode ( $\delta$ -CN).* In each DS,  $n_a$  is  $\delta$ -CN of the DS, if and only if  $|startpos - n_a| \leq \delta$ ;  $n_b$  is  $\delta$ -CN of the DS, if and only if  $|endpos - n_b| \leq \delta$ .

**Definition 7** *Dense Segment Set (DSS)*. A DSS consists of different DSs where the distance between adjacent DSs is not more than  $\delta$  and the total length of DSs in the DSS is not less than  $L$ , the density in the DSS is not less than  $\rho$ .

Actually, DSS may contain DSs located in different segments where DSs are joined by  $\delta$ -CN. DSS constitutes the road-network density query results. Suppose the density query parameter is given as  $(\rho, \delta, L, t_q)$ , where  $t_q$  is the query time. For query processing based on CUs, our algorithm includes two steps:

(1) **The filtering step:** Merge CUs into DSs by checking the parameter of  $\rho$  and  $\delta$ , which can prune some unnecessary segments. In this step, we can obtain a series of dense segments, specifically, a list of DSs and  $\delta$ -CNs.

(2) **The refinement step:** Merge the adjacent DSs around  $\delta$ -CNs to construct DSS by checking the parameter of  $\rho$ ,  $\delta$ ,  $L$  and finally find out the effective density query result consisting of dense segment sets.



**Fig. 2.** An example to construct DS and DSS

We explain the two steps of density query processing in detail. Firstly, according to network expansion approach [8], we traverse each segment to retrieve CUs sequentially, then compute the distance between adjacent CUs and the density of them. If the distance is not more than  $\delta$  and the density is not less than  $\rho$ , the CUs are merged to be a DS. Figure 2 shows an example. Given  $\rho=1.5$  and  $\delta=2$ , we compute  $DS$  at query time  $t_q$ . The road segment  $s_1$  (represented as  $\langle J_1, J_2 \rangle$ ) includes two CUs named  $cu_1$  and  $cu_2$ . Assume that the distance between  $cu_1$  and  $cu_2$  is 1.2 at  $t_q$  which is less than  $\delta$ , and the density is 1.8 after merging  $cu_1$  with  $cu_2$  which is more than  $\rho$ ,  $cu_1$  and  $cu_2$  can construct a DS (we call it  $DS_1$ ). The start position of  $DS_1$  is the head of  $cu_1$  and the end position of  $DS_1$  is the tail of  $cu_2$ . The number of objects in  $DS_1$  is the sum of the number of objects in  $cu_1$  and that in  $cu_2$ . Assume that the distance between  $DS_1$  and node  $J_2$  is 1.0 which is less than  $\delta$ ,  $J_2$  is the  $\delta$ -CN of  $DS_1$  (we call it  $\delta$ -CN<sub>1</sub>). We insert  $DS_1$  into the DS list of  $\delta$ -CN<sub>1</sub>. In this way, we can obtain  $DS_2$  on  $s_3$  including  $cu_4$  and  $DS_3$  on  $s_4$  including  $cu_3$  respectively. The  $\delta$ -CN of  $DS_2$  ( $\delta$ -CN<sub>2</sub>) is  $J_4$  and that of  $DS_3$  is  $J_2$ . So the DS list of  $\delta$ -CN<sub>1</sub> includes  $DS_1$  and  $DS_3$ , while the DS list of  $\delta$ -CN<sub>2</sub> includes  $DS_2$ . Algorithm 1 shows the pseudo.

In the refinement step, we compute dense segment sets so that the effective dense areas can be obtained. We traverse the list of each  $\delta$ -CN and evaluate whether those  $DS$ s around the  $\delta$ -CN can construct  $DSS$  based on the definition 8. Given  $L=100$  in Figure 2. As the  $\text{Distance}(DS_1, \delta\text{-CN}_1)=1.0$  and  $\text{Distance}(DS_3, \delta\text{-CN}_1)=0.7$ , the distance between  $DS_1$  and  $DS_3$  is 1.7, which is less than  $\delta$ . In addition, if  $DS_1$  is merged with  $DS_3$ , the density is more than  $\rho$ . Therefore,  $DS_1$  and  $DS_3$  can be merged to be a  $DSS$  named  $DSS_1$ . In the same way, we check if there are other dense segments can be merged with  $DSS_1$  by

---

**Algorithm 1:**  $Filter(\rho, \delta, t_q)$ 

---

**Input:** density threshold  $\rho$ , query time  $t_q$

```
1 begin
2   foreach  $e(n_x, n_y)$  of  $edgeList$  do
3     if  $e.cuList \neq null$  then
4       create a new DS:  $ds$ 
5        $cu \leftarrow getFirstCU(e)$ 
6        $ds.addCU(cu)$ ;  $ds.startpos = cu.pos$ 
7       if  $ds.startpos < \delta$  then
8          $ds.putCN(n_x)$ ;  $\delta-CN[n_x].putDS(ds)$ 
9       while  $getNextCU(e) \neq null$  do
10         $nextcu \leftarrow getNextCU(e)$ 
11        if  $Dd(ds, nextcu) > \delta$  or
12         $Dens(ds, nextcu) < \rho$  then
13           $ds.endpos = cu.pos + cu.len$ ;  $e.addDS(ds)$ 
14          create a new DS:  $ds$ 
15           $ds.startpos = nextcu.pos$ 
16           $ds.addCU(nextcu)$ ;  $cu = nextcu$ 
17         $ds.endpos = cu.pos + cu.len$ 
18        if  $1 - ds.endpos < \delta$  then
19           $ds.putCN(n_y)$ ;  $\delta-CN[n_y].putDS(ds)$ 
20         $e.addDS(ds)$ 
21 end
```

---

utilizing its  $\delta-CN$  and insert it into  $DSS_1$ . Finally, we check if the total length of  $DSS_1$  is more than  $L$ . If so,  $DSS_1$  is one of the answers of the density query. Repeat the process until all  $\delta-CN$ s containing dense segments are accessed. Then we can obtain all dense areas which are represented as dense segment sets at  $t_q$ . Note that a  $DS$  may be involved in the lists of two  $\delta-CN$ s. To avoid scanning the same nodes repeatedly, we mark the scanned  $\delta-CN$  as accessed node. Algorithm 2 shows the pseudo of the refinement step.

## 5 Performance Analysis

In this section, we first analyze the time complexity of the cluster-based query processing and then perform experimental evaluation.

### 5.1 Cluster-based Query Cost Analysis

Let  $n$  be the number of moving objects and  $m$  be the number of CUs created from the  $n$  objects at the initial time, where  $m \ll n$ . Thus, the average number of objects in a CU is  $n/m$ . Let  $W$  be the total number of edges and  $V$  be the total number of nodes of the road network. Let  $|E|$  denote the length of event priority queue  $E$ , which has a size of  $O(SE + ME)$ , where  $SE$  and  $ME$  are the number of split and merge events stored in  $E$ , respectively.

In the initial phase of our approach, the network is traversed and all the objects are grouped into CUs, and the initial split/merge events of all the CUs are calculated. It requires  $O(n + |V| \log |V|)$  time to build the initial CUs. Computing a split event from a CU takes  $O((n/m)^2)$  to predict the split time and inserting a split event to the priority queue  $E$  takes  $O(\log |E|)$ . Since there are  $m$  CUs, the

---

**Algorithm 2:** *Refinement*( $\rho, \delta, L, t_q$ )

---

**Input:** density threshold  $\rho$ , length threshold of DSS  $L$

**Output:** *Result*: The set of DSSs

```
1 begin
2   foreach  $\delta$ -CNi of  $\delta$ -CNList do
3     if ( $\delta$ -CNi.dsList  $\neq$  null) and (not  $\delta$ -CNi.accessed) then
4       /*Q is a priority queue to store all DSSs around  $\delta$ -CNi*/
5       /* $\delta$ -Q is a priority queue to store all unaccessed  $\delta$ -CNs*/
6       Q  $\leftarrow$  null;  $\delta$ -Q.put( $\delta$ -CNi)
7       while  $\delta$ -Q  $\neq$  null do
8         cn =  $\delta$ -Q.pop(); cn.accessed = true
9         Q.addDSS(cn); /*add all DSs around cn and sorted*/
10        create a new DSS: dss
11        ds = Q.pop(); dss.addDS(ds)
12         $\delta$ -Q.putdscn(ds); /*add all unaccessed  $\delta$ -CN around ds*/
13        while Q  $\neq$  null do
14          nextDS = Q.pop()
15          if Dist(dss, nextDS)  $\leq$   $\delta$  and Dens(dss, nextDS)  $\geq$   $\rho$  then
16            dss.addDS(nextDS)
17             $\delta$ -Q.putdscn(nextDS)
18        if dss.len  $>$  L then
19          Result.insert(dss)
20  return Result
21 end
```

---

cost of computing all the initial split events is  $O(m \log|E| + n^2/m)$ . To compute all the merge events and insert them into  $E$ , it requires  $O(2m \log|E|)$ , since there are  $2m$  pairs of adjacent CUs. Hence, the total time of the initial phase is  $O(n + |V| \log|V| + m \log|E| + n^2/m)$ .

The maintenance phase of CUs processes split/merge events from the priority queue  $E$ . For each split event that occurred on the edge, it takes  $O((n/m)^2)$  to compute the new split event for the CU and  $O(\log|E|)$  to insert this new split event to  $E$ . Therefore a split event on the edge is  $O(n^2/m^2 + m \log|E|)$  time complexity. For each split event at the node, the group split method needs  $O(n/m)$  to access and group the contained objects and  $O((n/m)^2)$  to compute the next split event for each split CU. The total cost to process a split in this case is  $O(n/m + n^2/m^2 + m \log|E|)$ .

For a merge event,  $O(\log|E|)$  is required to remove split/merge events associated with the two CUs. It takes  $O((n/m)^2)$  to compute the new split event and  $O(\log|E|)$  to insert the new split event to  $E$ . Finally, the merge events that are associated with the other CUs are recomputed and inserted into  $E$ . These operations require  $O(m \log|E|)$ . Hence, the total time of processing a merge event is  $O((n/m)^2 + m \log|E|)$ .

For query processing, in the filter step, we access the CUs in the whole network from each node by network expansion approach. In the refinement step, we only need to visit the  $\delta$ -CNs list and decrease the cost of traversing the network. Thus, the cost of query processing is  $O(|V| \log|V| + m)$ .

## 5.2 Experimental Evaluation

In this section, we compare the cluster-based density query processing with the existing density-based road-network clustering algorithm,  $\epsilon$ -link proposed by Yiu et al. [10] in terms of query performance and accuracy since  $\epsilon$ -link also returns the dense areas which consist of the density-based clusters of objects. We monitor the query results by running the  $\epsilon$ -link algorithm periodically and by maintaining CUs and finding the dense segments based on CUs.

**Experimental Settings** We implement the algorithms in C++ and carry out experiments on a Pentium 4, 2.4 GHz PC with 512MB RAM running Windows XP. For monitoring the dense areas in a road network, we design a moving object generator to produce synthetic datasets. The generator takes a map of a road network as input, and our experiment is based on the map of Beijing city. We set  $K$  hot spots (destination nodes) in the map. Initially, the generator places 80 percent objects around the hot spots and 20 percent objects at random positions, and updates their locations at each time unit. The query workload is 100 and each query has three parameters: (i) the density threshold  $\rho$ ; (ii) the threshold for dense segment length  $L$ ; (iii) the threshold for the distance of adjacent objects  $\delta$ . The query cost is measured in terms of CPU time. We also measure the accuracy of query answers. The parameters are summarized in Table 1, where values in bold denote the default values used.

**Table 1.** Parameters and their values

Parameter	Setting
Data size	100K, ..., 1M
Clustering threshold $\epsilon$	<b>0.5</b> , 1, ..., 3
Density threshold $\rho$	1, <b>1.5</b> , ..., 5.5
Dense segment threshold $\delta$	<b>2</b> , 2.5, 3, 3.5, 4
The total length threshold of dense segments $L$	<b>100</b> , 200, 300
Number of queries	100

**Comparison with the  $\epsilon$ -link Algorithm** To evaluate the performance, we first measure the total workload time and average query response time of two algorithms when varying the number of moving objects from 100K to 1M. We execute the CU maintenance and query processing in comparison with the static  $\epsilon$ -link on all objects at each time unit during 0 to 20 time units. For total workload time (shown in Figure 3), we measure the total CPU time including CU maintenance and query processing based on CUs. Figure 4 shows the average query response time for periodic query processing. In fact, considering the feature of road network, a CU represents the summary information of its objects and is incrementally updated over time with low cost, which can help speeding up the query processing. Therefore, our method is substantially better than the static one in terms of average query response time and still better in terms of total workload time.

**Accuracy Density Query** We evaluate the accuracy of density queries by computing average correct ratio of the number of objects in query result to that in the dataset around hot spots. Let *avgCorrectRate* represent the average correct ratio of query result, *Query\_objNum* be the number of objects of the

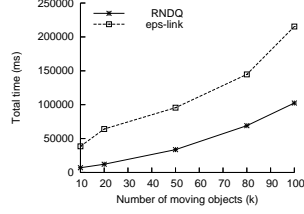


Fig. 3. Total time varies in data size

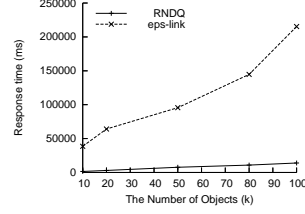


Fig. 4. Response time varies in data size

result,  $Real\_objNum$  be the average number of objects around each hot spot in the dataset,  $avgCorrectRate$  can be calculated by the following equation:

$$avgCorrectRate = \frac{1}{M} \sum_{i=1}^M \left(1 - \frac{|Query\_objNum - Real\_objNum|}{Real\_objNum}\right) \quad (1)$$

where  $M$  denotes the number of dense areas (i.e., DSS) in the query result. Figure 5 shows the comparison of the two methods in the query accuracy. We can see that the accuracy of our query algorithm is higher and stable with the different data distributions.

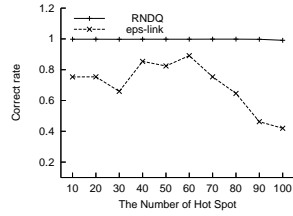


Fig. 5. Accuracy comparison

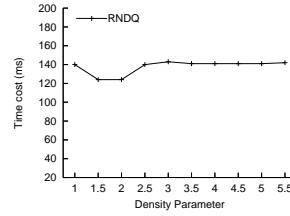


Fig. 6. Query Performance with  $\rho$

**Effect of Parameters** Finally, we study the effect of parameters ( $\rho$ ,  $L$ ,  $\delta$  and  $\epsilon$ ) of our methods on the query performance. Given  $\epsilon$  value at 2.5,  $\delta$  value at 4.5, and  $L$  value at 100, we change density threshold  $\rho$  to evaluate time cost of query processing. Figure 6 shows the experimental result. In addition, we also evaluate time cost by adjusting the parameter  $L$ , and the result is similar to Figure 6. Next, when fixing the  $\epsilon$  value at 2.5, we vary  $\delta$  to study its effect on the query processing. Finally, as the number of CUs depends on the system parameter  $\epsilon$ , we change the value of  $\epsilon$  from 0.5 to 3 to measure the maintenance cost of CUs. Figure 7 and Figure 8 show the effect of the two parameters. We observe that when  $\delta$  and  $\epsilon$  are set to 4.5 and 2.5, the method achieves the highest efficiency in our experimental settings.

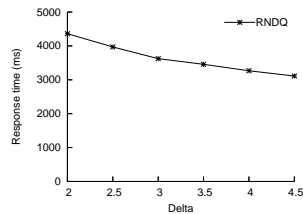


Fig. 7. Clustering performance with  $\delta$

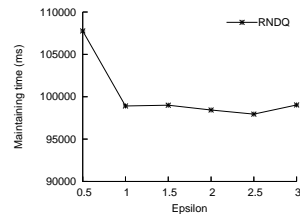


Fig. 8. Query performance with  $\epsilon$

## 6 Conclusion

In this paper, we introduce the definition of the dense segment and propose the problem of the effective road-network density query. Under our definition, we are able to answer queries for dense segments and find out dense areas in road network with arbitrary shape and arbitrary size. We present an cluster-based algorithm to response dynamic density queries and analyze the cost of cluster maintenance based on the object's movement feature in the road network. The cluster-based pre-processing can efficiently support density queries in road network. The experimental results show the efficiency and accuracy of our methods.

## References

1. Hyung-Ju Cho, Chin-Wan Chung: An Efficient and Scalable Approach to CNN Queries in a Road Network. VLDB 2005: 865-876.
2. Marios Hadjieleftheriou, George Kollios, Dimitrios Gunopulos, Vassilis J. Tsotras: On-Line Discovery of Dense Areas in Spatio-temporal Databases. SSTD 2003: 306-324
3. Anil K. Jain, Richard C. Dubes: Algorithms for Clustering Data. Prentice Hall, 1988
4. Christian S. Jensen, Dan Lin, Beng Chin Ooi, Rui Zhang: Effective Density Queries on Continuously Moving Objects. ICDE 2006: 71
5. George Karypis, Eui-Hong Han, Vipin Kumar: Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer, 1999, 32(8):68-75.
6. Yifan Li, Jiawei Han, Jiong Yang: Clustering moving objects. KDD 2004: 617-622
7. Kyriakos Mouratidis, Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis: Continuous Nearest Neighbor Monitoring in Road Networks. VLDB 2006: 43-54
8. Dimitris Papadias, Jun Zhang, Nikos Mamoulis, Yufei Tao: Query Processing in Spatial Network Databases. VLDB 2003: 802-813
9. Dragan Stojanovic, Slobodanka Djordjevic-Kajan, Apostolos N. Papadopoulos, Alexandros Nanopoulos: Continuous Range Query Processing for Network Constrained Mobile Objects. ICEIS (1) 2006: 63-70
10. Man Lung Yiu, Nikos Mamoulis: Clustering Objects on a Spatial Network. SIGMOD 2004: 443-454.
11. Man Lung Yiu, Nikos Mamoulis, Dimitris Papadias: Aggregate Nearest Neighbor Queries in Road Networks. IEEE Trans. Knowl. Data Eng. 17(6): 820-833 (2005)
12. Man Lung Yiu, Dimitris Papadias, Nikos Mamoulis, Yufei Tao: Reverse Nearest Neighbors in Large Graphs. IEEE Trans. Knowl. Data Eng. 18(4): 540-553 (2006)