

# Clustering Moving Objects in Spatial Networks

Jidong Chen<sup>1,2</sup>, Caifeng Lai<sup>1,2</sup>, Xiaofeng Meng<sup>1,2</sup>,  
Jianliang Xu<sup>3</sup>, and Haibo Hu<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  
{chenjd, laicf, xfmeng}@ruc.edu.cn

<sup>3</sup> Department of Computer Science, Hong Kong Baptist University  
{xujl, haibo}@comp.hkbu.edu.hk

**Abstract.** Advances in wireless networks and positioning technologies (e.g., GPS) have enabled new data management applications that monitor moving objects. In such new applications, realtime data analysis such as clustering analysis is becoming one of the most important requirements. In this paper, we present the problem of clustering moving objects in spatial networks and propose a unified framework to address this problem. Due to the innate feature of continuously changing positions of moving objects, the clustering results dynamically change. By exploiting the unique features of road networks, our framework first introduces a notion of cluster block (CB) as the underlying clustering unit. We then divide the clustering process into the continuous maintenance of CBs and periodical construction of clusters with different criteria based on CBs. The algorithms for efficiently maintaining and organizing the CBs to construct clusters are proposed. Extensive experimental results show that our clustering framework achieves high efficiency for clustering moving objects in real road networks.

**Keywords** Spatial-Temporal Databases, Moving Objects, Clustering, Spatial Networks

## 1 Introduction

Clustering is one of the most important analysis techniques. It groups similar data to provide a summary of data distribution patterns in a dataset. Early research mainly focused on clustering a static dataset [8, 11, 18, 3, 13, 6, 10, 4]. In recent years, clustering moving objects has been attracting increasing attention [9, 17, 7], which has various applications in the domains of weather forecast, traffic jam prediction, animal migration analysis, to name but a few. However, most existing work on clustering moving objects assumed a free movement space and defined the similarity between objects by their Euclidean distance.

In the real world, objects move within spatially constrained networks, e.g., vehicles move on road networks and trains on railway networks. Thus, it is more practical to define the similarity between objects by their network distance – the shortest path distance over the network. However, clustering moving objects in such networks is more complex than in free movement space. The increasing complexity first comes from the network distance metric. The distance between two arbitrary objects cannot be obtained in constant time, but requires an expensive shortest path computation. Moreover, the clustering results are related

to the segments of the network and their changes will be affected by the network constraint. For example, a cluster is likely to move along the road segments and change (i.e., split and merge) at the road junctions due to the objects' diversified spatio-temporal properties (e.g., moving in different directions). It is not efficient to predict their changes only by measuring their compactness. Thus, the existing clustering methods for free movement space cannot be applied to spatial networks efficiently.

On the other hand, the existing clustering algorithms based on the network distance [16] mainly focus on the static objects that lie on spatial networks. To extend to moving objects, we can apply them over the current positions of the objects in the network periodically. However, this approach is prohibitively costly since each time the expensive clustering evaluation starts from scratch. In addition, the clustering algorithms for different clustering criteria (e.g.,  $K$ -partitioning, distance, and density-based) are totally different in their implementation. This is inefficient for many applications that require to execute multiple clustering algorithms at the same time. For example, in a traffic management application, it is important to monitor those densely populated areas (by density-based clusters) so that traffic control can be applied; but at the same time, there may be a requirement for assigning  $K$  police officers to each of the congested areas. In this case, it is favorable to partition the objects into  $K$  clusters and keep track of the  $K$ -partitioned clusters. Separate evaluation of different types of clusters may incur computational redundancy.

In this paper, we propose a unified framework for "Clustering Moving Objects in spatial Networks" (CMON for short). The goals are to optimize the cost of clustering moving objects and support multiple types of clusters in a single application. The CMON framework divides the clustering process into the continuous maintenance of cluster blocks (CBs) and the periodical construction of clusters with different criteria based on CBs. A CB groups a set of objects on a road segment in close proximity to each other at present and in the near future. In general, a CB satisfies two basic requirements: 1) it is inexpensive to maintain in a spatial network setting; 2) it is able to serve as a building block of different types of application-level clusters. Our contributions are summarized as follows:

- We propose a unified framework for clustering moving objects in spatial networks to efficiently support different clustering criteria at the same time.
- We develop incremental CB maintenance (including split and merge) algorithms by analyzing the object movement features on a spatial network.
- We present efficient algorithms to periodically construct three kinds of clusters based on CBs. The network features are exploited to reduce the search space and avoid unnecessary computation of network distance.
- We show, through extensive experiments, that our clustering algorithms achieve high efficiency.

The rest of the paper is organized as follows. Section 2 surveys the related work. Section 3 describes the proposed framework. Section 4 details the initiation and maintenance of CBs. The algorithms for constructing the clusters with different clustering criteria based on CBs are proposed in Section 5. Section 6 shows experimental evaluations. We conclude this paper in Section 7.

## 2 Related Work

A lot of clustering techniques have been proposed for static datasets in a Euclidean space. They can be classified into the partitioning [8, 11], hierarchical [18, 3, 13], density-based [10], grid-based [15, 1], and model-based [2] clustering methods. There are also a few studies [4, 6, 16] on clustering nodes or objects in a spatial network. Yiu and Mamoulis [16] defined the problem of clustering objects based on the network distance, which is mostly related to our work. They proposed algorithms for three different clustering paradigms, i.e., *k-medoids* for *K*-partitioning,  *$\epsilon$ -link* for density-based, and *single-link* for hierarchical clustering. These algorithms avoid computing distances between every pair of network nodes by exploiting the properties of the network. However, all these solutions assumed a static dataset. As discussed in the Introduction, a straightforward extension of these algorithms to moving objects by periodical re-evaluation is inefficient. Besides, Jin *et al.* [5] studied the problem of mining distance-based outliers in spatial networks, but it is only a byproduct of clustering.

Clustering analysis on moving objects has recently drawn increasing attentions. Li *et al.* [9] first addressed this problem by proposing a concept of micro moving cluster (MMC), which denotes a group of similar objects both at current time and at near future time. Each MMC maintains a bounding box for the moving objects contained, whose size grows over time. Even the CB in our framework is some kind of micro-cluster, it has much differences from MMC. First MMC is based on the Euclidean distance metric while CB is formed by the network distance. Second, MMC does not consider the network constraint where micro-clusters usually move along the road segment with the objects and change at the road junctions immediately. The prediction of the MMC's split and merge in a spatial network is therefore not accurate. The bounding boxes of MMCs are likely to be exceeded frequently and numbers of maintenance events dominate the overall running time of the algorithms. Finally, as the detailed object information in a MMC is not maintained, it can only support very limited clustering paradigms. While CB uses the distance of neighboring objects to measure the compactness instead of the boundary objects of micro-cluster, it is therefore capable to construct global clusters with different criteria. Afterwards, Zhang and Lin [17] proposed a histogram construction technique based on a clustering paradigm. In [7], Kalnis proposed three algorithms to discover moving clusters from historical trajectories of objects. Nehme and Rundensteiner [12] applied the idea of clustering moving objects to optimize the continuous spatio-temporal query execution. The moving cluster is represented by a circle in their algorithms. However, most above works only considered moving objects in unconstrained environments and defined the similarity between objects by their Euclidean distance. To the best of our knowledge, this is the first work which specifies on the problem of clustering network-constrained moving objects whose similarity is defined by network distance.

## 3 The System Model and CMON Framework

In this section, we describe the system model and present a unified CMON framework for clustering moving objects in a spatial network. It aims to optimize the cost of clustering evaluation and support clusters with different criteria.

We model a spatial network as a graph where objects are moving on the edges [16] (we use the segments interchangeably in this paper). The distance between any two objects, called *network distance*, is measured by the length of the shortest path connecting them in the network. We employ a similar motion model as in [9], where moving objects are assumed to move in a piecewise linear manner (i.e., each object moves at a stable velocity at each edge). We assume that an object location update has the following form  $(oid, n_a, n_b, pos, speed, next\_node)$ , where *oid* is the id of the moving object,  $(n_a, n_b)$  represents the edge on which the object moves (from  $n_a$  towards  $n_b$ ), *pos* is the relative location to  $n_a$ , and *speed* is the moving speed. We also assume that the next edge to move along,  $(n_b, next\_node)$ , is known in advance. The requirement is to continuously monitor the moving clusters with various criteria at some predefined period.

As shown in Figure 1, the proposed CMON framework is composed of two components: the incremental maintenance of cluster blocks (CBs) and the periodical construction of different types of application-level clusters. A CB is a group of moving objects close to each other at present and near future time. For easy maintenance, we constrain the objects in a CB moving in the same direction and on the same edge segment. Additionally, a CB imposes a strict clustering criterion so as to support different types of application-level clusters. Specifically, the network distance between each pair of neighboring objects in a CB does not exceed a preset threshold  $\epsilon$ . Formally, a CB is defined as follows:

**Definition 1** *Cluster Block.* A cluster block is represented by  $CB = (O, n_a, n_b, head, tail, ObjNum)$ , where  $O$  is a list of objects  $\{o_1, o_2, \dots, o_i, \dots, o_n\}$ ,  $o_i = (oid_i, n_a, n_b, pos_i, speed_i, next\_node_i)$ . 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 edge  $(n_a, n_b)$ , the position of the cluster is determined by an interval  $(head, tail)$  in terms of the network distance from  $n_a$ . Thus, the length of the CB is  $|tail - head|$ . *ObjNum* is the number of objects in the CB.

Note that the edge, position, length, and object number of a CB appear as its summary information. We incrementally maintain each CB by taking into account the objects' anticipated movements. We capture the predicted update events (including split and merge events) of each CB during the continuous movement and process these events accordingly (see Section 4 for details). At any time, clusters of different criteria can be constructed from the CBs, instead of the entire set of moving objects, which makes the construction processing cost efficient. Moreover, to reduce unnecessary computation of the network distance between the CBs, we adapt the network expansion method to combine CBs to construct the application-level clusters (see Section 5 for details).

## 4 Maintenance of CBs

Initially, based on the CB definition, a set of CBs are created by traversing all edge segments in the network and their associated objects. The CBs are incrementally maintained after their creation. As time elapses, the distance between adjacent objects in a CB may exceed  $\epsilon$  and, hence, we need to split the CB. A CB may also merge with adjacent CBs when they are within the distance of  $\epsilon$ . Thus, for each CB, we predict the time when they may split or merge. The predicted split

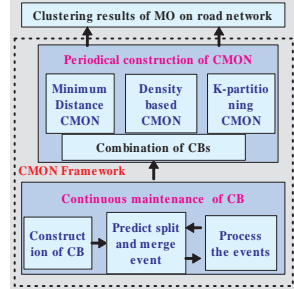


Fig. 1. CMON Framework

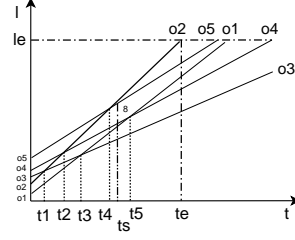


Fig. 2. Prediction of Splitting CB

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 (compute) the split and merge events for affected CBs (new CBs if any). This process is continuously repeated. The key problems are: 1) how to predict split/merge time of a CB, and 2) how to process a split/merge event of a CB.

The split of a CB may occur in two cases. The first is when CB arriving at the end of the segment (i.e., an intersection node of the spatial network). When the moving objects in a CB reach an intersection node, the CB has to be split since they may head in different directions. Obviously, a split time is the time when the first object in the CB arrives at the node. In the second case, the split of a CB is when the distance between some neighboring objects moving on the segment exceeds  $\epsilon$ . However, it is not easy to predict the split time since the neighborhood of objects changes over time. And 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 CB 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 the CB. Otherwise, we update the order of objects starting from  $t_m$  and repeat the same process until some distance exceeds  $\epsilon$  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 CB.

Figure 2 shows an example. Given  $\epsilon = 7$ , we compute the split time as follows. At the initial time  $t_0$ , the CB is formed with a list of objects  $\langle o_1, o_2, o_3, o_4, o_5 \rangle$ . We first compute the time  $t_e$  when the first object (i.e.,  $o_2$ ) arrives at the end of the segment (i.e.,  $le$ ). For adjacent objects, we find that the earliest meeting time is  $t_1$  at which  $o_2$  and  $o_3$  first meet. We then compare the maximum distance for each pair of adjacent objects during  $[t_0, t_1]$  and no pair whose distance exceeds 7. At  $t_1$ , the object list is updated into  $\langle o_1, o_3, o_2, o_4, o_5 \rangle$ . In the same way, the next meeting time is at  $t_2$  for  $o_2$  and  $o_4$ . There is also no neighboring objects whose distance exceeds 7 during  $[t_1, t_2]$ . As the algorithm continues, at  $t_4$ , the object list becomes  $\langle o_3, o_1, o_4, o_5, o_2 \rangle$  and  $t_5$  is the next time for  $o_1$  and  $o_4$  to meet. When comparing neighboring objects during  $[t_4, t_5]$ , we find the  $o_4$  and  $o_5$  whose distance is longer than 7 at time  $t_s$ . Since  $t_s < t_e$ , we obtain  $t_s$  as the split time of the CB.

We now discuss how to handle a split event. If the split event happens on the segment, we can simply split the CB into two ones and predict the split and merge events for each of them. If the split event occurs at the end of the segment, the processing would be more complex. One straightforward method is

to handle the departure of the objects individually each time an object reaches the end of the segment. Obviously, the cost of this method is high. To reduce the processing cost, we propose a group split scheme. When the first object leaves the segment, we split the original CB into several new CBs according to objects' directions (which can be implied from *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 CB and retain the CB 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 CBs. Only valid objects will be counted in constructing application-level clusters. Figure 3 illustrates this split example. When  $CB_1$  reaches  $J_1$ , objects  $p_1$  and  $p_3$  will move to the segment  $\langle J_1, J_2 \rangle$  while  $p_2$  and  $p_4$  will follow  $\langle J_1, J_6 \rangle$ . Thus,  $CB_1$  is split into two such that  $p_2$  and  $p_4$  join  $CB_3$ , and  $p_1$  and  $p_3$  form a new cluster  $CB_4$ . We still keep  $CB_1$  until  $p_4$  leaves  $\langle J_4, J_1 \rangle$ . As can be seen, the group split scheme reduces the number of split events and hence the cost of CB maintenance.

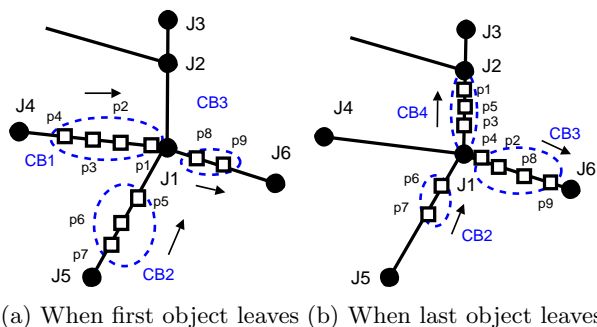


Fig. 3. Group Split at an Edge Intersection

The merge of CBs may occur when adjacent CBs in a segment are moving together (i.e. their network distance  $\leq \epsilon$ ). To predict the initial merge time of CBs, we dynamically maintain the boundary objects of each CB and their validity time (the period when they are treated as boundary of the CB), and compare the minimum distances between the boundary objects of two CBs with the threshold  $\epsilon$  at their validity time. The boundary objects of CBs can be obtained by maintaining the order of objects during computing the split time. For the example in Figure 2, the boundary objects of the CB are represented by  $(o_1, o_5)$  for validity time  $[t_0, t_3]$ ,  $(o_3, o_5)$  for  $[t_3, t_4]$ , and  $(o_3, o_2)$  for  $[t_4, t_e]$ . 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 CBs into one CB and compute the split time and merge time of the merged CB. Finally, the corresponding affected CBs in the event queue are updated.


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

## 5 CMON Construction with Different Criteria

This section discusses how to construct application-level clusters of different criteria from CBs. We focus our discussions on three common clustering criteria, i.e., distance-based, density-based, and  $K$ -partitioning.

### 5.1 Distance-based CMON

A common clustering criterion is based on the minimum distance metric. The Minimum Distance CMON is defined as follows:

**Definition 2** *Minimum Distance CMON (MD-CMON).* For each object in an MD-CMON, the minimum network distance with other objects in the cluster is not longer than a user specified threshold  $\delta$  ( $\delta \geq \epsilon$ ). 

The requirement of  $\epsilon \leq \delta$  is necessary because it guarantees that a CB does not cross two clusters in the MD-CMON. The MD-CMON can be constructed by combining the CBs. Generally, for two CBs, we need to compute their network distance (i.e., the minimum network distance of their boundary objects) to determine whether to combine them. This simple method has a time complexity of  $O(N^2)$ , where  $N$  is the number of CBs. In order to reduce the computation cost, we adapt the incremental network expansion method to combine the CBs. The detailed algorithm can be found in Algorithm 1.

The algorithm starts with a CB and adds its adjacent nodes that are within  $\delta$  to a queue  $Q$  using Dijkstra's algorithm. Take Figure 4 as an example. Suppose  $\delta = 10$  and the algorithm starts with  $CB_1$ . Thus, initially  $CB_1$  is marked "visited" and  $J_1$  is added to  $Q$ . The algorithm proceeds to dequeue the first node in  $Q$  (i.e.,  $J_1$ ). All adjacent edges of  $J_1$  (except the checked edge  $\langle J_6, J_1 \rangle$ ) are examined. For each edge  $\langle J_1, J_i \rangle$ , assuming  $dist(J_1, J_i)$  to be the edge length, if  $J_i$  satisfies  $dist(CB_1, J_1) + dist(J_1, J_i) \leq \delta$ ,  $J_i$  is added to  $Q$  and  $dist(CB_1, J_i) = dist(CB_1, J_1) + dist(J_1, J_i)$ . Moreover, all unvisited CBs on each adjacent edge are checked. For a  $CB_i$  on  $\langle J_1, J_i \rangle$ , if  $dist(CB_1, J_1) + dist(J_1, CB_i) \leq \delta$ ,  $CB_i$  is merged into  $CB_1$ 's MD-CMON cluster. If  $dist(CB_i, J_i) \leq \delta$  and  $J_i$  has not been added to  $Q$ , it is added to  $Q$ . The algorithm continues with the same process until  $Q$  becomes empty and the CBs around  $CB_1$  are combined into a cluster  $C_1$ . Afterwards, the algorithm picks up another unvisited CB and repeats the same process until all CBs are visited.

### 5.2 Density-based CMON

The second clustering criterion is density-based, which is good at filtering out noise data.

**Definition 3** *Density-based CMON (DB-CMON).* For each cluster in the DB-CMON, the average density should be higher than a given threshold  $\rho$ . Moreover, there should not be any empty segment (without any objects lying on) whose length is longer than  $E$ .

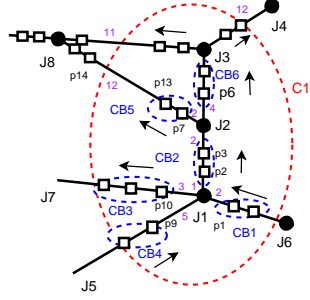


Fig. 4. The Combination of CBs

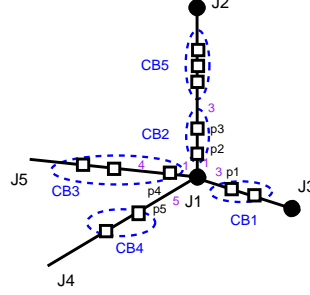


Fig. 5. The Cross-CB

**Algorithm 1: MD\_CMON()**


---

```

1  foreach  $CB_i$  do
2    if  $CB_i.visited == false$  then
3       $Q =$  new priority queue;
4      find edge  $n_x, n_y$  where  $CB_i$  lies;
5       $CB = CB_i; C = CB;$ 
6       $nextCB =$  Next CB on  $n_x, n_y$  from  $CB_i$  to  $n_y$ ;
7      while ( $nextCB \neq null$ ) and  $Dist(CB.head, nextCB.tail) \leq \delta$  do
Merge_Expand( $CB, nextCB, C, n_x, n_y$ );
8      if ( $nextCB == null$ ) and  $Dist(CB.head, n_y) \leq \delta$  then
9         $B.node = n_y; B.dist = Dist(CB.head, n_y);$ 
10       Enqueue( $Q, B$ );
11     while notempty( $Q$ ) do
12        $B =$  Dequeue( $Q$ );
13       foreach node  $n_z$  adjacent to  $B.node$  do
14          $nextCB =$  Next CB from  $B.node$  to  $n_z$ ;
15         if ( $nextCB \neq null$ ) and  $Dist(B.node, nextCB.tail) + B.dist \leq \delta$ 
then
16            $newd_{n_z} = Dist(nextCB.head, n_z);$ 
17           Merge_Expand( $CB, nextCB, C, B.node, n_z$ );
18           while ( $nextCB \neq null$ ) and
19              $Dist(CB.head, nextCB.tail) \leq \delta$  do
20                $newd_{n_z} = Dist(nextCB.head, n_z);$ 
21               Merge_Expand( $CB, nextCB, C, B.node, n_z$ );
22           if (no CBs on edge ( $B.node, n_z$ )) then
23              $newd_{n_z} = B.dist + Dist(B.node, n_z);$ 
24           if ( $nextCB == null$ ) and ( $newd_{n_z} \leq \delta$ ) then
25              $B_{new}.node = n_z; B_{new}.dist = newd_{n_z};$ 
           Enqueue( $Q, B_{new}$ );

```

---



---

**Procedure Merge\_Expand**( $CB_1, CB_2, C, node_1, node_2$ )

---

```

1 if  $CB_2.visited == false$  then
2    $C = MergeClst(C, CB_2)$ ;
3    $CB_1 = CB_2$ ;  $CB_1.visited = true$ ;
4    $CB_2 = Next\ CB\ from\ node_1\ to\ node_2$ ;
5 else
6    $C_1 = FindCluster(CB_2)$ ;
7    $C = MergeClst(C, C_1)$ ;

```

---

Suppose there are  $m$  ( $m > 1$ ) objects in a CB, we have the density of the CB as  $\frac{m}{\epsilon(m-1)} > \frac{1}{\epsilon}$ . The second condition is necessary to avoid very skewed clusters. It is equivalent to the condition that for any object in the cluster, the nearest object is within a distance of  $E$ . Thus, to construct the DB-CMON clusters from CBs, we require  $\epsilon \leq \max\{E, \frac{1}{\rho}\}$ .

The cluster formation algorithm is the same as the one described in Algorithm 1 except that the minimum-distance constraint (transformed from the density constraint) is dynamic. Suppose the density of the current cluster with  $k$  objects is  $\rho'$  and a CB has  $m$  objects with a length of  $L$ . If a CB can be merged into the cluster, their minimum distance  $D$  must satisfy  $\frac{k+m}{k/\rho'+L+D} \geq \rho$ , i.e.,  $D \leq \frac{k+m+\rho(k/\rho'+L)}{\rho}$ .

### 5.3 K-Partitioning CMON

K-Partitioning CMON is similar to the K-Partitioning clustering method [8, 11]. It can be defined as follows:

**Definition 4** *K-Partitioning CMON (KP-CMON)*. Given a set of objects, group them into  $K$  clusters such that the sum of distances between all adjacent objects in each cluster is minimized.

According to the definition of CBs, the sum of distances between all adjacent objects in each CB is minimized. Therefore, it is intuitive to construct the KP-CMON clusters from the CBs. An exhaustive method is to iteratively combine the closest pairs of CBs until  $K$  clusters are obtained. This method requires to compute the distances between all pairs of CBs, which is costly. Hereby, we propose a low-complexity heuristic similar to the  $K$ -means algorithm [8, 11]. We initially select  $K$  CBs as the seeds for  $K$  clusters. For the remaining CBs, we assign them to their nearest clusters to make the sum of distances between adjacent objects to be minimum. Note that this heuristic may not lead to the optimal solution. Suppose that in Figure 5, the distances between CBs are:  $dist(CB_2, CB_3) < dist(CB_2, CB_5) < dist(CB_3, CB_1) < dist(CB_2, CB_1) < dist(CB_3, CB_5)$ , and that the initial seed CBs are  $CB_1$  and  $CB_5$  for  $K = 2$ . When  $CB_3$  is checked, it will be assigned to the cluster of  $\{CB_1\}$ . Then  $CB_2$  will be assigned to the cluster of  $\{CB_5\}$ , which is different from the optimal solution where  $CB_2$  and  $CB_3$  should be grouped together since  $dist(CB_2, CB_3) < dist(CB_2, CB_5)$ . To compensate for such mistakes, we introduce the concept of Cross-Cluster for adjacent CBs lie around the same node, if their minimum distance is less than  $\epsilon$ , we

group them into a Cross-CB. Then, the clustering algorithm is applied over the CBs and Cross-CBs.

## 6 Performance Analysis

In this section, we evaluate the performance of our proposed techniques by comparing with the periodical clustering. We implement the CMON algorithms in C++ and carry out experiments on a Pentium 4, 2.4 GHz PC with 512MB RAM running Windows XP. Our performance study uses synthetic datasets. For monitoring the effective clusters in the road network, we design a moving object generator. 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 in the map. Initially, the generator places eighty percent objects around the hot spots and twenty percent objects at random positions, and updates their locations at each time unit. Each object around the hot spot moves along its shortest path from its initial hot spot to another one. We compare the MD-CMON algorithm against the  $\epsilon$ -link algorithm proposed in [16]. We monitor the clustering results by running the  $\epsilon$ -link algorithm periodically and by maintaining the CBs created at the initial time and combining them to construct the MD-CMON.

First, we compare our method with the static  $\epsilon$ -link by measuring both average clustering response time and total workload time when varying the number of moving objects from 100K to 1M. We set the clustering frequency at 1 per time unit and execute the CBs maintenance and combination in comparison with the static  $\epsilon$ -link on all objects. For total workload time (shown in Figure 6), we measure the total CPU time including maintaining CB and combining CBs to clusters up to 20 time units. Figure 7 also shows the average clustering response time for periodic clustering requests. In essence, CBs are like  $B^+$ -tree or R-tree index for periodical queries and they share the same property, i.e., amortizing the query (clustering) cost to maintain the data structure (CBs) for speeding up the query (clustering) processing. Therefore, our method is substantially better than the static one in terms of average response time, yet is still better in terms of total workload time.

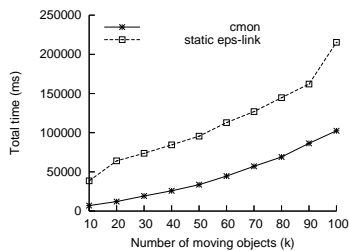


Fig. 6. Total time varies in data size

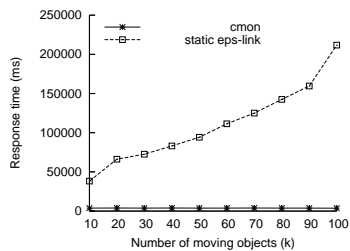


Fig. 7. Response time varies in data size

Then, we change the clustering frequency with  $1/5$ ,  $1/4$ , ...,  $1$  to examine how the total time is affected. The experiment is executed on 100K moving objects during 20 time units. Figure 8 shows the results of the two methods under

different clustering frequencies. We can see that the higher the clustering frequency, the more efficient our CMON method. In addition, we fix the clustering frequency at 1 and measure the clustering response time at different clustering monitoring instances. As time elapses, the objects change their locations continuously, which may affect the clustering efficiency. As shown in Figure 9, our CMON method consistently keeps a lower cost than the static  $\epsilon$ -link method over different time instances.

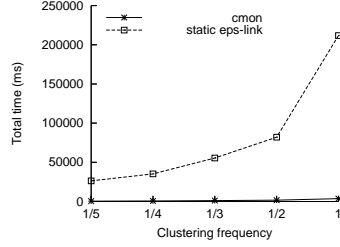


Fig. 8. Clustering frequency effect

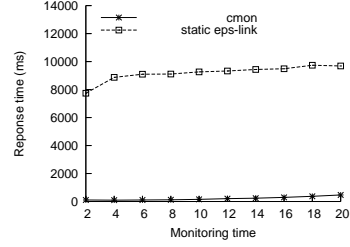


Fig. 9. Monitoring time effect

Finally, we study the effect of parameters ( $\epsilon$  and  $\delta$ ) of our methods on the clustering efficiency. As the number of CBs depends on the system parameter  $\epsilon$ , we change the value of  $\epsilon$  from 0.5 to 3 to measure the maintenance cost of CBs. Then when fixing the  $\epsilon$  value at 2.5, we vary  $\delta$  to study its effect on the CB combination to clusters. Figure 10 and Figure 11 show the effect of the two parameters. We observe that when  $\epsilon$  and  $\delta$  are set to 2.5 and 4.5, the method achieves the highest efficiency in our experimental setting.

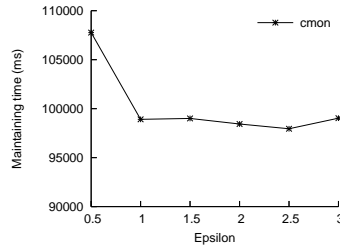


Fig. 10. CMON performance with  $\epsilon$

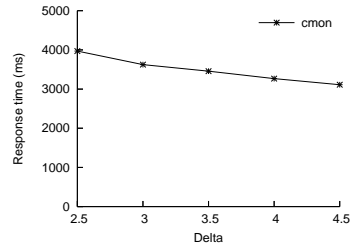


Fig. 11. CMON performance with  $\delta$

## 7 Conclusion

In this paper, we studied the problem of clustering moving objects in a spatial road network and proposed a framework to address this problem. By introducing a notion of cluster block, this framework, on one hand, amortizes the cost of

clustering into CB maintenance and combination based on the object movement feature in the road network; and on the other hand, it efficiently supports different clustering criteria. We exploited the features of the road network to predict the split and merge of CBs accurately and efficiently. Three different clustering criteria have been defined and the cluster construction algorithms based on CBs were proposed. The experimental results showed the efficiency of our method.

## Acknowledgments

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); Program for Creative PhD Thesis in University. Jianliang Xu's work was supported by grants from the Research Grants Council, Hong Kong SAR, China (Project Nos. HKBU 2115/05E and HKBU 2112/06E).

## References

1. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan: Automatic subspace clustering of high dimensional data for data mining applications. SIGMOD 1998: 94-105.
2. D. Fisher: Knowledge acquisition via incremental conceptual clustering. Machine Learning, 1987, 2:139-172.
3. S. Guha, R. Rastogi, and K. Shim: CURE: An efficient clustering algorithm for large databases. SIGMOD 1998: 73-84.
4. A. K. Jain and R. C. Dubes: Algorithms for Clustering Data. Prentice Hall, 1988.
5. W. Jin, Y. Jiang, W. Qian, A. K. H. Tung: Mining Outliers in Spatial Networks. DASFAA 2006: 156-170.
6. G. Karypis, E. H. Han, and V. Kumar: Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer, 1999, 32(8):68-75.
7. P. Kalnis, N. Mamoulis, S. Bakiras: On Discovering Moving Clusters in Spatio-temporal Data. SSTD 2005: 364-381.
8. L. Kaufman and P. J. Rousseeuw: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley and Sons, Inc, 1990.
9. Y.F. Li, J.W. Han, J. Yang: Clustering Moving Objects. KDD 2004: 617-622.
10. E. Martin, H. P. Kriegel, J. Sander, and X. Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. SIGKDD 1996: 226-231.
11. R. T. Ng and J. Han: Efficient and effective clustering methods for spatial data mining. VLDB 1994: 144-155.
12. R. V. Nehme, E. A. Rundensteiner: SCUBA: Scalable Cluster-Based Algorithm for Evaluating Continuous Spatio-temporal Queries on Moving Objects. EDBT 2006: 1001-1019.
13. A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos: C2P: Clustering based on closest pairs. VLDB 2001: 331-340.
14. D. Papadias, J. Zhang, N. Mamoulis, Y. Tao: Query Processing in Spatial Network Databases. VLDB 2003: 790-801.
15. W. Wang, Yang, R. Muntz, STING: A Statistical Information grid Approach to Spatial Data Mining. VLDB 1997: 186-195.
16. M. L. Yiu, N. Mamoulis: Clustering Objects on a Spatial Network. SIGMOD 2004: 443-454.
17. Q. Zhang, X. Lin: Clustering Moving Objects for Spatio-temporal Selectivity Estimation. ADC 2004: 123-130.
18. T. Zhang, R. Ramakrishnan, and M. Livny: BIRCH: An efficient data clustering method for very large databases. SIGMOD 1996: 103-114.