

# Continuous Clustering of Moving Objects

Christian S. Jensen, Dan Lin, Beng Chin Ooi

**Abstract**—This paper considers the problem of efficiently maintaining a clustering of a dynamic set of data points that move continuously in two-dimensional Euclidean space. This problem has received little attention and introduces new challenges to clustering. The paper proposes a new scheme that is capable of incrementally clustering moving objects. This proposal employs a notion of object dissimilarity that considers object movement across a period of time, and it employs clustering features that can be maintained efficiently in incremental fashion. In the proposed scheme, a quality measure for incremental clusters is used for identifying clusters that are not compact enough after certain insertions and deletions. An extensive experimental study shows that the new scheme performs significantly faster than traditional ones that frequently rebuild clusters. The study also shows that the new scheme is effective in preserving the quality of moving-object clusters.

**Index Terms**—Spatial databases, Temporal databases, Clustering

## I. INTRODUCTION

In abstract terms, clustering denotes the grouping of a set of data items so that similar data items are in the same groups and different data items are placed in distinct groups. Clustering thus constitutes fundamental data analysis functionality that provides a summary of data distribution patterns and correlations in a dataset. Clustering is finding application in diverse areas such as image processing, data compression, pattern recognition, and market research, and many specific clustering techniques have been proposed for static datasets (e.g., [17], [28]).

With the increasing diffusion of wireless devices such as PDAs and mobile phones and the availability of geo-positioning, e.g., GPS, a variety of location-based services are emerging. Many such services may exploit knowledge of object movement for purposes such as targeted sales, system load-balancing, and traffic congestion prediction [3]. The needs for analyses of the movements of a population of objects have also been fueled by natural phenomena such as cloud movement and animal migration. However, in spite of extensive research having been conducted on clustering and on moving objects (e.g., [12], [15], [20], [21], [24]), little attention has been devoted to the clustering of moving objects.

A straightforward approach to the clustering of a large set of continuously moving objects is to do so *periodically*. However, if the period is short, this approach is overly expensive, mainly because the effort expended on previous clustering are not leveraged. If the period is long, long durations of time exist with no clustering information available. Moreover, this brute-force approach effectively treats the objects as static object and does not take into account the information about their movement. For example, this has the implication that it is impossible to detect that some groups of data are moving together.

Rather, clustering of continuously moving objects should take into account not just the objects' current positions, but also their anticipated movements. As we shall see, doing so enables us to

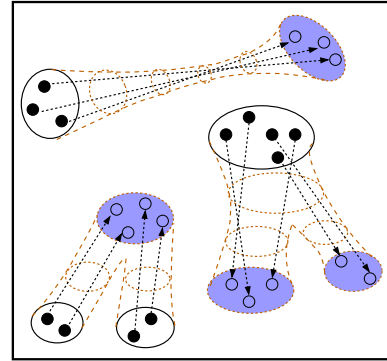


Fig. 1. Clustering of Moving Objects

capture each clustering change as it occurs during the continuous motion process, thus providing better insight into the clustering of datasets of continuously moving objects. Figure 1 illustrates the clustering effect that we aim for. Connected black and the white points denote object positions at the current time and a near-future time. Our approach attempts to identify clusters at the current time, as given by solid ellipses, and to detect cluster splits and merges at future times, as represented by shaded ellipses.

As has been observed in the literature, two alternatives exist when developing a new incremental clustering scheme [18]. One is to develop an entirely new, specialized scheme for the new problem of moving objects. The other is to utilize the framework provided by a standard clustering algorithm, but to develop new summary data structures for the specific problem being addressed that may be maintained efficiently in incremental fashion and that may be integrated into such a framework. We adopt this second alternative, as we believe that this is more flexible and generic. In particular, the new summary data structures may then be used together with a broad range of existing standard clustering algorithms. In addition, the summary data structures can be used for other data mining tasks such as computing approximate statistics of datasets.

We consequently propose a new summary data structure, termed a *clustering feature*, for each moving object cluster, which is able to reflect key properties of a moving cluster and can be maintained incrementally. Based on these clustering features, we modify the Birch algorithm [28] to enable moving object clustering. As suggested, our scheme can also be applied to other incremental clustering algorithms based on cluster centers.

We summarize our contributions as follows. We employ a notion of object dissimilarity that considers object movement across a period of time. We develop clustering features that can be maintained incrementally in efficient fashion. In our scheme, a quality measure for incremental clusters is proposed to identify clusters that are not compact enough after certain insertions and deletions. In other words, we are able to predict when clusters are to be split, thus avoiding the handling of the large

amounts of events akin to the bounding-box violations of other methods [16]. An extensive experimental study shows that the proposed scheme performs significantly faster than traditional schemes that frequently rebuild clusters. The results also show that the new scheme is effective in preserving the quality of clusters of moving objects. To the best of our knowledge, this is the first disk-based clustering method for moving objects.

The organization of the paper is as follows. Section II reviews related work. Section III presents our clustering scheme. Section IV covers analytical studies, and Section V reports on empirical performance studies. Finally, Section VI concludes the paper.

## II. RELATED WORK

Many clustering techniques have been proposed for static data sets [1], [2], [7], [10], [14], [17], [18], [19], [25], [28]. A comprehensive survey is given elsewhere [11]. The K-means algorithm [17] and the Birch algorithm [28] are representatives of non-hierarchical and hierarchical methods, respectively. The goal of the K-means algorithm is to divide the objects into  $K$  clusters such that some metric relative to the centroids of the clusters is minimized. The Birch algorithm, which is proposed to incrementally cluster static objects, introduces the notion of a clustering feature and a height-balanced clustering feature tree. Our approach extends these concepts. A key difference is that while in Birch, summary information of static data does not need to be changed unless an object is inserted, in our approach, the summary information itself must be dynamic and must evolve with time due to continuous object movement.

Another interesting clustering algorithm is due to Yiu and Mamoulis [26], who define and solve the problem of object clustering according to network distance. In their assumed setting, where objects are constrained to a spatial network, network distance is more realistic than the widely used Euclidean distance for the measurement of similarity between objects.

In spite of extensive work on the static databases, only few approaches exist for moving-object clustering. We proceed to review each of these.

Early work by Har-Peled [9] aims to show that moving objects can be clustered once so that the resulting clusters are competitive at any future time during the motion. However, in two-dimensional space, the static clusters obtained from this method may have about 8 times larger radii than the radii obtained by the optimal clustering, and the numbers of clusters are also much larger (at least 15 times) than for the usual clustering. Further, this proposal does not take into account I/O efficiency.

Zhang and Lin [27] propose a histogram technique based on the clustering paradigm. In particular, using a “distance” function that combines both position and velocity differences, they employ the K-center clustering algorithm [6] for histogram construction. However, histogram maintenance lacks in efficiency—as stated in the paper, a histogram must be reconstructed if too many updates occur. Since there are usually a large amount of updates at each timestamp in moving object databases, the histogram reconstruction will occur frequently and thus this approach may not be feasible.

Li et al. [16] apply micro-clustering [28] to moving objects, thus obtaining algorithms that dynamically maintain bounding boxes of clusters. However, the numbers of maintenance events involved dominates the overall running times of the algorithms,

and the numbers of such events are usually prohibitively large. Given a moving micro-cluster that contains  $n$  objects, the objects at each edge of the bounding box can change up to  $O(n)$  times during the motion, and each change corresponds to an event.

Kalnis et al. [13] study historical trajectories of moving objects, proposing algorithms that discover moving clusters. A moving cluster is a sequence of spatial clusters that appear in consecutive snapshots of the object movements, so that consecutive spatial clusters share a large number of common objects. Such moving clusters can be identified by comparing clusters at consecutive snapshots; however, the comparison cost can be very high. More recently, Spiliopoulou et al. [22] propose a framework MONIC which models and traces cluster transitions. Specifically, they first cluster data at multiple timestamps by using the bisecting K-means algorithm, and then detect the changes of clusters at different timestamps. Unlike the above two works, which analyze the relations between clusters after the clusters are obtained, our proposal aims to predict the possible cluster evolution to guide the clustering.

Finally, we note that clustering of moving objects involves future-position modeling. In addition to the linear function model, which is used in most work, a recent proposal considers non-linear object movement [23]. The idea is to derive a recursive motion function that predicts the future positions of a moving object based on the positions in the recent past. However, this approach is much more complex than the widely adopted linear model and complicates the analysis of several interesting spatio-temporal problems. Thus, we use the linear model. We also note that we have been unable to find work on clustering in the literature devoted to kinetic data structures (e.g., [4]).

## III. MOVING-OBJECT CLUSTERING

This section first describes the representation of moving objects, then proposes a scheme to cluster moving objects, called Moving-Object Clustering (MC for short).

### A. Modeling of Moving Objects

We assume a population of moving objects, where each object is capable of transmitting its current location and velocity to a central server. An object transmits new movement information to the server when the deviation between its current, actual location and its current, server-side location exceeds a specified threshold, dictated by the services to be supported. The deviation between the actual location and the location assumed by the server tends to increase as time progresses.

In keeping with this, we define the *maximum update time* ( $U$ ) as a problem parameter that denotes the maximum time duration in-between any two updates to any object. Parameter  $U$  can be built into the system to require that each object must issue at least one update every  $U$  time units. This is rational due to the concern that if an object did not communicate with the server for a long time, it is hard to know whether this object keeps moving in the same way or disappears accidentally without being able to notify the server.

Each moving object has a unique ID, and we model its point position in two-dimensional Euclidean space as a linear function of time. Specifically, an object with ID  $OID$  can be represented by a four-tuple  $(OID, \bar{x}_u, \bar{v}, t_u)$ , where  $\bar{x}_u$  is the position of the object at time  $t_u$  and  $\bar{v}$  is the velocity of the object at that time.

Then the (server-side) position of this object at time  $t$  can be computed as  $\bar{x}(t) = \bar{x}_u + \bar{v}(t - t_u)$ , where  $t \geq t_u$ .

### B. Object Movement Dissimilarity

We aim to cluster objects with similar movements, taking into account both their initial position and velocity. In particular, we use weighted object positions at a series of time points to define object dissimilarity. The computation of dissimilarity proceeds in three steps.

We first select  $m$ ,  $m \geq 1$ , sample timestamps  $t_1, \dots, t_m$ , each of which is associated with a weight  $w_i$ . Their properties are described as follows, where  $t_{now}$  denotes the current time:

$$\forall i (t_i < t_{i+1} \wedge t_{now} \leq t_i \leq t_{now} + U \wedge w_i \geq w_{i+1})$$

We thus only consider trajectories of moving objects within a period of duration  $U$  after the current time, and sample points are given higher weight the closer they are to the current time. This allows modeling of predicted positions that become less accurate as time passes. The details of the selection of weight values follow in Section IV.

In the second step, object positions are computed at the chosen timestamps according to their movement functions. Given an object  $O$ , its positions at times  $t_1, \dots, t_m$  are  $\bar{x}^{(1)}, \dots, \bar{x}^{(m)}$ . The Euclidean distance (ED) between a pair of positions  $\bar{x}_1^{(i)}$  and  $\bar{x}_2^{(i)}$  of two objects  $O_1$  and  $O_2$  at time  $t_i$  is given by  $ED(\bar{x}_1^{(i)}, \bar{x}_2^{(i)}) = |\bar{x}_1^{(i)} - \bar{x}_2^{(i)}| = \sqrt{(x_{11}^i - x_{21}^i)^2 + (x_{12}^i - x_{22}^i)^2}$ , where  $x_{jk}^i$  is the  $k$ th dimensional position value of object  $O_j$  at time  $t_i$ .

Third, we define the dissimilarity function between  $O_1$  and  $O_2$ :

$$M(O_1, O_2) = \sum_{i=1}^m w_i \cdot ED^2(\bar{x}_1^{(i)}, \bar{x}_2^{(i)}) \quad (1)$$

Note that when  $m = 1$  and  $w_1 = 1$ , the function reduces to the (squared) Euclidean distance.

We extend the function to apply to an object and a cluster  $C$  that consists of  $N$  objects and has center  $O_c$ :

$$M(O, C) = \frac{N}{N+1} \sum_{i=1}^m w_i \cdot ED^2(\bar{x}^{(i)}, \bar{x}_c^{(i)}) \quad (2)$$

The center  $O_c$  of a cluster is defined formally in the following section.

### C. Clustering Feature

We proceed to define the clustering feature for moving objects, which is a compact, incrementally maintainable data structure that summarizes a cluster and that can be used for computing the average radius of a cluster.

**Definition 1:** The *clustering feature* (CF) of a cluster is of the form  $(N, \overline{CX}, \overline{CX^2}, \overline{CV}, \overline{CV^2}, \overline{CXV}, t)$ , where  $N$  is the number of moving objects in the cluster,  $\overline{CX} = \sum_{i=1}^N \bar{x}_i(t)$ ,  $\overline{CX^2} = \sum_{i=1}^N \bar{x}_i^2(t)$ ,  $\overline{CV} = \sum_{i=1}^N \bar{v}_i(t)$ ,  $\overline{CV^2} = \sum_{i=1}^N \bar{v}_i^2(t)$ ,  $\overline{CXV} = \sum_{i=1}^N (\bar{x}_i(t)\bar{v}_i(t))$ , and  $t$  is the update time of the feature.

A clustering feature can be maintained incrementally under the passage of time and updates.

**Claim 1:** Let  $t_{now}$  be the current time and  $CF = (N, \overline{CX}, \overline{CX^2}, \overline{CV}, \overline{CV^2}, \overline{CXV}, t)$ , where  $t < t_{now}$ , be a clustering feature. Then  $CF$  at time  $t$  can be updated to  $CF'$  at time  $t_{now}$  as follows:

$$CF' = \left( N, \overline{CX} + \overline{CV}(t_{now} - t), \frac{\overline{CX^2} + 2\overline{CXV}(t_{now} - t) + \overline{CV^2}(t_{now} - t)^2}{\overline{CV}, \overline{CV^2}, \overline{CXV} + \overline{CV^2}(t_{now} - t), t_{now}} \right).$$

*Proof:* The number of moving objects  $N$ , the sum of the velocities  $\overline{CV}$ , and the sum of the squared velocities  $\overline{CV^2}$  remain the same when there are no updates. The three components that involve positions need to be updated to the current time according to the moving function. For example,  $\overline{CX}$  will be updated to  $\overline{CX}'$  as follows.

$$\begin{aligned} \overline{CX}' &= \sum_{i=1}^N \bar{x}_i(t_{now}) \\ &= \sum_{i=1}^N (\bar{x}_i(t) + \bar{v}_i(t_{now} - t)) \\ &= \sum_{i=1}^N \bar{x}_i(t) + (t_{now} - t) \sum_{i=1}^N \bar{v}_i \\ &= \overline{CX} + \overline{CV}(t_{now} - t) \end{aligned}$$

The other two components are derived similarly.  $\square$

**Claim 2:** Assume that an object given by  $(OID, \bar{x}, \bar{v}, t)$  is inserted into or deleted from a cluster with clustering feature  $CF = (N, \overline{CX}, \overline{CX^2}, \overline{CV}, \overline{CV^2}, \overline{CXV}, t)$ . The resulting clustering feature  $CF'$  is computed as:  $CF' = (N \pm 1, \overline{CX} \pm \bar{x}, \overline{CX^2} \pm \bar{x}^2, \overline{CV} \pm \bar{v}, \overline{CV^2} \pm \bar{v}^2, \overline{CXV} \pm \bar{x}\bar{v}, t)$ .

*Proof:* Omitted.  $\square$

**Definition 2:** Given a cluster  $C$ , its (virtual, moving) center object  $O_c$  is  $(OID, \overline{CX}/N, \overline{CV}/N, t)$ , where the  $OID$  is generated by the system.

This center object represents the moving trend of the cluster.

**Definition 3:** The *average radius*  $R(t)$  of a cluster is the time-varying average distance between the member objects and the center object. We term  $R(t)$  the average-radius function.

$$R(t) = \sqrt{\frac{1}{N} \sum_{i=1}^N ED^2(\bar{x}_i(t), \bar{x}_c(t))}$$

This function enables us to measure the compactness of a cluster, which then allows us to determine when a cluster should be split. More importantly, we can efficiently compute the time when a cluster needs to be split without tracking the variation of the bounding box of the cluster.

**Claim 3:** The average-radius function  $R(t_2)$  can be expressed as a function of time,  $R(\Delta t)$ , and can be computed based on the clustering feature given at time  $t_1$  ( $t_1 \leq t_2$ ).

*Proof:* Let the clustering feature be given as of time  $t_1$  and assume that we want to compute  $R(t_2)$  for a later time  $t_2$ . We first substitute the time variation  $\Delta t = t_2 - t_1$  for every occurrence of  $t_2 - t_1$  in function  $R(t_2)$ .

$$\begin{aligned} ED^2(\bar{x}_i(t), \bar{x}_c(t)) &= \sum_{i=1}^N (\bar{x}_i(t_2) - \bar{x}_c(t_2))^2 \\ &= \sum_{i=1}^N (\bar{x}_i^2(t_2) - 2\bar{x}_i(t_2)\bar{x}_c(t_2) + \bar{x}_c^2(t_2)) \\ &= \sum_{i=1}^N ((\bar{x}_i + \bar{v}_i\Delta t)^2 - 2(\bar{x}_i + \bar{v}_i\Delta t)(\bar{x}_c + \bar{v}_c\Delta t) + (\bar{x}_c + \bar{v}_c\Delta t)^2) \end{aligned}$$

Then we represent function  $R(t_2)$  as a function of  $\Delta t$ :

$$R(\Delta t) = \sqrt{(A\Delta t^2 + B\Delta t + C)/N} \quad \text{where}$$

$$A = \sum_{i=1}^N \bar{v}_i^2 - 2\bar{v}_c \sum_{i=1}^N \bar{v}_i + N\bar{v}_c^2$$

$$B = 2\left(\sum_{i=1}^N (\bar{x}_i\bar{v}_i) - \bar{v}_c \sum_{i=1}^N \bar{x}_i - \bar{x}_c \sum_{i=1}^N \bar{v}_i + N\bar{x}_c\bar{v}_c\right)$$

$$C = \sum_{i=1}^N \bar{x}_i^2 - 2\bar{x}_c \sum_{i=1}^N \bar{x}_i + N\bar{x}_c^2$$

Subsequently, the coefficients of function  $\Delta t$  can be expressed in terms of the clustering feature.

$$\begin{aligned} A &= \overline{CV^2} - (\overline{CV})^2/N \\ B &= 2(\overline{CXV} - \overline{CX}\overline{CV}/N) \\ C &= \overline{CX^2} - (\overline{CX})^2/N \quad \square \end{aligned}$$

#### D. Clustering Scheme

We are now ready to present our clustering scheme, which employs the proposed dissimilarity function and clustering feature, thus enabling many traditional incremental clustering algorithms based on cluster centers, to handle moving objects.

Our scheme utilizes the framework provided by the Birch clustering algorithm, which, however, requires several modifications and extensions: (i) concerning the data structure, we introduce two auxiliary data structures in addition to the hierarchical data structure; (ii) we propose algorithms for the maintenance of the new clustering feature under insertion and deletion operations; (iii) for the split and merge operations, we propose algorithms that quantify the cluster quality and compute the split time.

1) *Data Structures*: The clustering algorithm uses a disk-based data structure that consists of directory nodes and cluster nodes. The directory nodes store summary information for the clusters. Each node contains entries of the form  $\langle CF, CP \rangle$ , where  $CF$  is the clustering feature and  $CP$  is a pointer to either a cluster node or the next directory node. The structure allows the clusters to be organized hierarchically according to the center objects of the clusters, and hence is scalable with respect to data size. The directory node size is one disk page.

Each cluster node stores the data objects, each represented as  $(OID, \bar{x}, \bar{v}, t)$ , according to the cluster they belong to. Unlike the directory node, each cluster node may consist of multiple disk pages. The maximum capacity of a cluster is an application dependent parameter, which can be given by users. By using the concept of maximum cluster capacity, we guarantee that the clustering performance is stable, i.e., the maintenance cost for each cluster is similar. It should be noted that the maximum cluster capacity is only associated with the leaf cluster nodes. The nodes at higher levels correspond to bigger clusters and can also be returned to the users according to their requests.

In addition to this clustering feature structure, two auxiliary structures, an event queue and a hash table, are also employed. The event queue stores future split events  $\langle t_{split}, CID \rangle$  in ascending order of  $t_{split}$ , where  $t_{split}$  denotes the split time and  $CID$  is the cluster identifier. The hash table maps object IDs to cluster IDs, i.e.,  $OIDs$  to  $CIDs$ , so that given the ID of an object, we can efficiently locate the cluster that this object belongs to. These two structures store much less data than the whole dataset (the event queue and the hash table are only 1% and 10% of the whole data set size, respectively), and hence they can be either cached in main memory or stored contiguously on disk for efficient scanning and loading into main memory.

2) *Insertion and Deletion*: We proceed to present the algorithms that maintain a clustering under insertions and deletions.

The outline of the insertion algorithm is given in Figure 2. To insert an object  $O$  given by  $(OID, \bar{x}, \bar{v}, t_u)$ , we first find the center object of some cluster  $C$  that is nearest to the object according to  $M$ . A *global partition threshold*  $\rho_g$  is introduced

#### Insert ( $O$ )

Input:  $O$  is an object to be inserted

1. find the nearest center object  $O_c$  of  $O$   
//  $O_c$  belongs to cluster  $CID$
  2. **if**  $M(O_c, O) > \rho_g$  **then**
  3.     create a new cluster for  $O$
  4. **else**
  5.      $t_s \leftarrow \text{SplitTime}(CID, O)$
  6.     **if**  $t_s$  is not equal to the current time  
       and cluster  $CID$  is not full **then**
  7.         insert  $O$  into cluster  $CID$
  8.         adjust the clustering feature of cluster  $CID$
  9.         **if**  $t_s > 0$  **then**
  10.          insert event  $(t_s, CID)$  into the event queue
  11.          insert  $O$  to the hash table
  12.     **else**
  13.         split( $CID, O, newCID$ )
  14.         **if** CanMerge( $CID, CID_1$ )
  15.             **then** merge( $CID, CID_1$ )
  16.         **if** CanMerge( $newCID, CID_2$ )
  17.             **then** merge( $newCID, CID_2$ )
- end Insert.

Fig. 2. Insertion Algorithm

that controls the clustering. Threshold  $\rho_g$  gives the possible maximum  $M$  distance between two objects belonging to two closest neighboring clusters. To estimate  $\rho_g$ , we first need to know the average size of a cluster  $S_c$ . Without any prior knowledge,  $S_c$  is computed as  $S_c = \text{Area}/(N/f)$  based on a uniform distribution ( $\text{Area}$  is the area of the domain space,  $N$  is the total number of objects, and  $f$  is the cluster capacity). If the data distribution is known,  $\text{Area}$  can be computed as the area of the region covered by most objects.

We can now define  $\rho_g = \sum_{i=1}^m w_i \cdot (2\sqrt{S_c})^2$ . The idea underlying this definition is that if the distance between two objects is always twice as large as the average cluster diameter during the considered time period, these two objects most possibly belong to two different clusters. By using  $\rho_g$ , we can roughly partition the space, which saves computation cost. If the distance between object  $O$  and cluster  $C$  exceeds  $\rho_g$ , we create a new cluster for object  $O$  directly. Otherwise, we check whether cluster  $C$  needs to be split after absorbing object  $O$ . If no split is needed, we insert object  $O$  into cluster  $C$  and then execute the following adjustments.

- Update the clustering feature of  $C$  to the current time, according to Claim 1; then update it to cover the new object, according to Claim 2.
- Calculate the split time, if any, of the new cluster and insert the event into the event queue. Details to do with splits are addressed in the next section.
- Update the object information in the hash table.

If cluster  $C$  is to be split after the insertion of object  $O$ , we check whether the two resultant clusters ( $CID$  and  $newCID$ ) can be merged with other clusters. The function CanMerge may return a candidate cluster for merge operation. Specifically, an invocation of function CanMerge with arguments  $CID$  and  $CID'$ , looks for a cluster that it is appropriate to merge cluster  $CID$  with, and if such a cluster is found, it is returned as  $CID'$ . The merge policy will be explained in Section III-D.3.

Next, to delete an object  $O$ , we use the hash table to locate the cluster  $C$  that object  $O$  belongs to. Then we remove object  $O$

**Delete ( $O$ )**

Input:  $O$  is an object to be deleted

1.  $CID = Hash(O)$   
// object  $O$  belongs to cluster  $CID$
  2. delete  $O$  from the hash table
  3. delete  $O$  from cluster  $CID$
  4. adjust the clustering feature of cluster  $CID$
  5. **if** cluster  $CID$  is in underflow
  6.     **if**  $CanMerge(CID, CID')$
  7.         **then**  $merge(CID, CID')$
  8.     **else**
  9.         delete old event of cluster  $CID$  from the event queue
  10.         insert new event of cluster  $CID$  into the event queue
- end Delete.

Fig. 3. Deletion Algorithm

from the hash table and cluster  $C$ , and we adjust the clustering feature. Specifically, we first update the feature to the current time according to Claim 1 and then modify it according to Claim 2. If cluster  $C$  does not underflow after the deletion, we further check whether the split event of  $C$  has been affected and adjust the event queue accordingly. Otherwise, we apply the merge policy to determine whether this cluster  $C$  can be merged with other clusters (denoted as  $CID'$ ). The deletion algorithm is outlined in Figure 3.

3) *Split and Merge of Clusters*: Two situations exist where a cluster must be split. The first occurs when the number of objects in the cluster exceeds a user-specified threshold (i.e., the maximum cluster capacity). This situation is detected automatically by the insertion algorithm covered already. The second occurs when the average radius of the cluster exceeds a threshold, which means that the cluster is not compact enough. Here, the threshold (denoted as  $\rho_s$ ) can be defined by the users if they want to limit the cluster size. It can also be estimated as the average radius of clusters given by the equation  $\rho_s = \frac{1}{4}\sqrt{S_c}$ . We proceed to address the operations in the second situation in some detail.

Recall that the average radius of a cluster is given as a function of time  $R(\Delta t)$  (cf. Section III-C). Since  $R(\Delta t)$  is a square root, for simplicity, we consider  $R^2(\Delta t)$  in the following computation. Generally,  $R^2(\Delta t)$  is a quadratic function. It degenerates to a linear function when all the objects have the same velocities. Moreover,  $R^2(\Delta t)$  is either a parabola opening upwards or an increasing line—the radius of a cluster will never first increase and then decrease when there are no updates. Figure 4 shows the only two cases possible for the evolution of the average radius when no updates occur, where the shaded area corresponds to the region covered by the cluster as time passes.

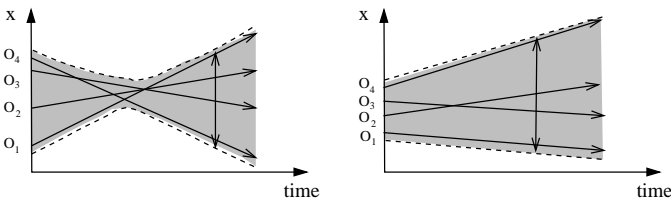


Fig. 4. Average Radius Examples

Our task is to determine the time, if any, in-between the current time and the maximum update time when the cluster must be split, i.e.,  $\Delta t$  ranges from 0 to  $U$ . Given the split threshold  $\rho_s$ , three

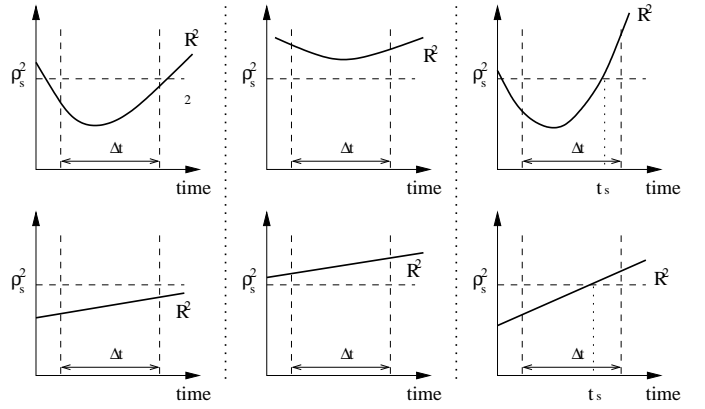


Fig. 5. Squared Average Radius Evolution

kinds of relationships between  $R^2(\Delta t)$  and  $\rho_s^2$  are possible—see Figure 5.

In the first, leftmost two cases, radius  $R^2$  remains below threshold  $\rho_s^2$ , implying that no split is caused. In the second, middle two cases, radius  $R^2(0)$  exceeds threshold  $\rho_s^2$ , which means that the insertion of a new object into cluster  $CID$  will make the new radius larger than the split threshold and thus cause an immediate split. In the last two cases, radius  $R^2$  exceeds threshold  $\rho_s^2$  at time  $t_s$ , causing an event  $\langle t_s, CID \rangle$  to be placed in the event queue.

The next step is to identify each of the three situations by means of function  $R^2(\Delta t)$  itself. We first compute  $R^2(0)$ . If this value exceeds  $\rho_s^2$ , we are in the second case. Otherwise,  $R^2(U)$  is computed. If this value is smaller than  $\rho_s^2$ , we are in the first case. If not, we are in the third case, and we need to solve the equation  $(A\Delta t^2 + B\Delta t + C)/N = \rho_s^2$ , where the split time  $t_s$  is the larger solution, i.e.,  $t_s = (-B + \sqrt{B^2 - 4A(C - \rho_s^2 N)}) / (2A)$ . Note that when the coefficient of  $\Delta t^2$  equals 0, function  $R^2(\Delta t)$  degenerates to a linear function and  $t_s = (\rho_s^2 N - C) / B$ . Figure 6 summarizes the algorithm.

At the time of a split, the split starts by identifying the pair of objects with the largest  $M$  value. Then, we use these objects as seeds, redistributing the remaining objects among them, again based on their mutual  $M$  values. Objects are thus assigned to the cluster that they are most similar to. We use this splitting procedure mainly because it is very fast and running time is an important concern in moving object environments. The details of

**SplitTime ( $CID, O$ )**

Input: Cluster  $CID$  and object  $O$

Output: The time to split the cluster  $CID$  with  $O$

1. get function  $R(t)$  from the cluster  $CID$  and  $O$
  2. **if**  $R^2(0) > \rho_s^2$  **then**
  3.     **return** current time  
// need to split at the current time
  4. **else**
  5.     **if**  $R^2(U) \leq \rho_s^2$  **then**
  6.         **return** -1 // no need to split during  $U$
  7.     **else**
  8.         compute the split time  $t_s$  by  $R^2(t_s) = \rho_s^2$
  9.     **return**  $t_s$  // return the future split time
- end SplitTime.

Fig. 6. Split Time Algorithm

---

**Split** ( $CID_1, O, CID_2$ )  
Input: Cluster  $CID_1$  and object  $O$   
Output: New cluster with ID  $CID_2$

1. pick the farthest pair of objects ( $seed_1, seed_2$ )  
from cluster  $CID_1$  and  $O$  based on  $M$
2. initialize cluster  $CID_2$
3. insert  $seed_2$  into cluster  $CID_2$
4. delete  $seed_2$  from cluster  $CID_1$
5. **for** each remaining object  $O_r$  in  $CID_1$  **do**
6.      $D_{m1} \leftarrow M(O_r, seed_1)$
7.      $D_{m2} \leftarrow M(O_r, seed_2)$
8.     **if**  $D_{m1} > D_{m2}$  **then**
9.         insert  $O_r$  into cluster  $CID_2$
10.         modify the hash table
11.         **if**  $O_r$  belongs to cluster  $CID_1$  **then**
12.             delete  $O_r$  from cluster  $CID_1$
13. adjust the clustering feature of cluster  $CID_1$
14. compute the clustering feature of cluster  $CID_2$
15. **return**  $CID_2$

end Split.

---

Fig. 7. Split Algorithm

the algorithm are shown in Figure 7.

We first pick up the farthest pair of objects  $seed_1$  and  $seed_2$  (line 1), which will be stored in cluster  $CID_1$  and  $CID_2$  respectively. For each remaining object  $O_r$  in cluster  $CID_1$ , we compute its distances to  $seed_1$  and  $seed_2$  using  $M$  (lines 6–7). If  $O_r$  is close to  $seed_1$ , it will remain in cluster  $CID_1$ . Otherwise,  $O_r$  will be stored in cluster  $CID_2$ . After all the objects have been considered, we compute the clustering features of both clusters (lines 11–12).

After a split, we check whether each cluster  $C$  among the two new clusters can be merged with preexisting clusters (see Figure 8). To do this, we compute the  $M$ -distances between the center object of cluster  $C$  and the center object of each preexisting cluster. We consider the  $k$  nearest clusters that may accommodate cluster  $C$  in terms of numbers of objects. For each such candidate, we execute a “virtual merge” that computes the clustering feature assuming absorption of  $C$ . This allows us to identify clusters where the new average radius is within threshold  $\rho_g$ . Among

---

**CanMerge**( $CID_1, CID_2$ )  
Input: Cluster  $CID_1$ , waiting for a merge operation  
Output: Cluster  $CID_2$ , a candidate for a merge operation

1. **for** each cluster  $CID_x$  except  $CID_1$  **do**
2.     **if** cluster  $CID_x$  has enough space to absorb cluster  $CID_1$
3.         **then**
4.              $D_m \leftarrow M(O_x, O_1)$   
//  $O_x$  is the center object of cluster  $CID_x$   
//  $O_1$  is the center object of cluster  $CID_1$
5.             update list  $L_c$  that records the  $k$  nearest clusters
6.     **for** each cluster  $CID_2$  in  $L_c$  **do**
7.          $CF \leftarrow CF(CID_2) + CF(CID_1)$
8.         compute possible split time  $t_s$  from  $CF$
9.         **if**  $t_s < 0$  **then** // no need to split
10.             **return**  $CID_2$
11.         **else**
12.             record  $CID_2$  with the largest  $t_s$
13. **return**  $CID_2$

end CanMerge.

---

Fig. 8. Identifying Clusters to be Merged

---

**Merge**( $CID_1, CID_2$ )  
Input: Cluster  $CID_1$  and  $CID_2$  to be merged

1.  $CF_1 \leftarrow CF(CID_1)$  at the current time
2.  $CF_2 \leftarrow CF(CID_2)$  at the current time
3.  $CF_1 \leftarrow CF_1 + CF_2$
4. **for** each object  $O$  in cluster  $CID_2$  **do**
5.     store  $O$  in cluster  $CID_1$
6.     update the hash table
7. delete cluster  $CID_2$
8. delete split event of cluster  $CID_2$  from event queue
9. compute split time  $t_s$  of new cluster  $CID_1$
10. modify split event of cluster  $CID_1$  in event queue

end Merge.

---

Fig. 9. Merge Algorithm

these, we choose a cluster that will lead to no split during the maximum update time, if one exist; otherwise, we choose the one that will yield the latest split time. Finally, we execute the real merge: we update the clustering feature, the hash table, and the event queue. The merge algorithm is shown in Figure 9.

#### IV. ANALYSIS OF DISSIMILARITY VERSUS CLUSTERING

In this section, we study the relationship between dissimilarity measure  $M$  and the average radius of the clusters produced by our scheme.

To facilitate the analysis, we initially assume that no updates occur to the dataset. This enables us to set the weights used in  $M$  to 1—decreasing weights are used to make later positions, which may be updated before they are reached, less important. Also to facilitate the analysis, we replace the sum of sample positions in  $M$  with the corresponding integral, denoted as  $M'$ , from the time when a clustering is performed and  $U$  time units into the future. Note that  $M'$  is the boundary case of  $M$  that is similar to the integrals used in R-tree based moving object indexing [21].

The next theorem states that inclusion of an object into the cluster with a smaller  $M'$  value leads to a tighter and thus better clustering during time interval  $U$ .

*Theorem 1:* Let  $O = (OID, \bar{x}, \bar{v}, t_u)$  denote an object to be inserted at time  $t_u$ ;  $C_i$ ,  $i = 1, 2$ , denote two existing clusters with  $N_i$  objects, center objects  $O_{ci} = (OID_{ci}, \bar{x}_{ci}, \bar{v}_{ci}, t_u)$ , and average radii  $R_i$  at time  $t_u$ . Let  $R_{i,O}$  be the average radius of  $C_i$  after absorbing object  $O$ . If  $M'(O, C_1) < M'(O, C_2)$  then the average squared distance between objects and cluster centers after inserting  $O$  to cluster  $C_1$  is less than that after inserting  $O$  to cluster  $C_2$ :

$$\int_0^U \frac{(N_1 + 1)R_{1,O}^2 + N_2R_2^2}{N_1 + N_2 + 1} dt < \int_0^U \frac{N_1R_1^2 + (N_2 + 1)R_{2,O}^2}{N_1 + N_2 + 1} dt.$$

*Proof:*  $M'(O, C_i)$  computes the difference between the position of object  $O$  and the center object  $O_{ci}$  of cluster  $C_i$  for the  $U$  time units starting at the insertion time  $t_u$ . Let  $\bar{x}(t)$  and  $\bar{x}_{ci}(t)$  denote the positions of objects  $O$  and  $O_{ci}$  at time  $t_u + t$ . We first reorganize  $M'$  to be function of the time  $t$  that ranges from 0 to  $U$ .

$$\begin{aligned} M'(O, C_i) &= \frac{N_i}{N_i + 1} \int_0^U [\bar{x}(t) - \bar{x}_{ci}(t)]^2 dt \\ &= \frac{N_i}{N_i + 1} \int_0^U [(\bar{x} + \bar{v}t) - (\bar{x}_{ci} + \bar{v}_{ci}t)]^2 dt \\ &= \frac{N_i}{N_i + 1} \left[ \frac{1}{3}(\bar{v} - \bar{v}_{ci})^2 U^3 + (\bar{x} - \bar{x}_{ci})(\bar{v} - \bar{v}_{ci})U^2 \right. \\ &\quad \left. + (\bar{x} - \bar{x}_{ci})^2 U \right] \end{aligned}$$

Next, we examine the variation of the radius of the cluster that absorbs the new object  $O$ .

$$\begin{aligned} & \int_0^U (N_i + 1)R_{i,O}^2 dt - \int_0^U N_i R_i^2 dt \\ &= \int_0^U [(N_i + 1) \frac{(A_{i,O}t^2 + B_{i,O}t + C_{i,O})}{N_i + 1} - N_i \frac{(A_i t^2 + B_i t + C_i)}{N_i}] dt \\ &= \int_0^U [(A_{i,O} - A_i)t^2 + (B_{i,O} - B_i)t + (C_{i,O} - C_i)] dt \\ &= \frac{1}{3}(A_{i,O} - A_i)U^3 + \frac{1}{2}(B_{i,O} - B_i)U^2 + (C_{i,O} - C_i)U \end{aligned}$$

We proceed to utilize Theorem 3, which states that the average radius of a cluster can be computed from the cluster's clustering feature. In the transformation from the third to the fourth line, we use  $\overline{CV}_i = N_i \bar{v}_{ci}$ .

$$\begin{aligned} \Delta A_i &= A_{i,O} - A_i \\ &= (\overline{CV}_i^2 + \bar{v}^2 - \frac{(\overline{CV}_i + \bar{v})^2}{N_i + 1}) - (\overline{CV}_i^2 - \frac{\overline{CV}_i^2}{N_i}) \\ &= \frac{1}{N_i(N_i + 1)} \overline{CV}_i^2 - \frac{2}{N_i + 1} \overline{CV}_i \bar{v} + \frac{N_i}{N_i + 1} \bar{v}^2 \\ &= \frac{N_i}{N_i + 1} \bar{v}^2 + \frac{N_i^2}{N_i(N_i + 1)} \bar{v}_{ci}^2 - \frac{2N_i}{N_i + 1} \bar{v}_{ci} \bar{v} \\ &= \frac{N_i}{N_i + 1} (\bar{v} - \bar{v}_{ci})^2 \end{aligned}$$

We express  $\Delta B_i$  similarly. In the last transformation, we use  $\overline{CX}_i = N_i \bar{x}_{ci}$  and  $\overline{CX}_i = N_i \bar{x}_{ci}$ .

$$\begin{aligned} \Delta B_i &= B_{i,O} - B_i \\ &= 2(\overline{CX}_i + \bar{x}) - \frac{(\overline{CX}_i + \bar{x})(\overline{CV}_i + \bar{v})}{N_i} - \\ &\quad 2(\overline{CX}_i - \frac{\overline{CX}_i \overline{CV}_i}{N_i}) \\ &= \frac{2N_i}{N_i + 1} (\bar{x} - \bar{x}_{ci})(\bar{v} - \bar{v}_{ci}) \end{aligned}$$

Finally, we express  $\Delta C_i$ , utilizing  $\overline{CX}_i = N_i \bar{x}_{ci}$ .

$$\begin{aligned} \Delta C_i &= C_{i,O} - C_i \\ &= (\overline{CX}_i^2 + \bar{x}^2 - \frac{(\overline{CX}_i + \bar{x})^2}{N_i + 1}) - (\overline{CX}_i^2 - \frac{\overline{CX}_i^2}{N_i}) \\ &= \frac{N_i}{N_i + 1} (\bar{x} - \bar{x}_{ci})^2 \end{aligned}$$

We observe that  $M'(O, C_i) = \int_0^U (N_i + 1)R_{i,O}^2 dt - \int_0^U N_i R_i^2 dt$ . Utilizing the premise of the theorem, we have  $\int_0^U (N_1 + 1)R_{1,O}^2 dt - \int_0^U N_1 R_1^2 dt < \int_0^U (N_2 + 1)R_{2,O}^2 dt - \int_0^U N_2 R_2^2 dt$ . Then, both sides of the inequality are divided by the total number of objects in  $C_1$  and  $C_2$ , which is  $N_1 + N_2 + 1$ . The theorem follows by rearranging the terms.  $\square$

The following lemma, based on Theorem 1, shows which cluster a new object should be inserted into.

*Lemma 2:* Placement of a new object into the cluster  $C$  with the nearest center object according to dissimilarity measure  $M$  minimizes the average squared distance between all objects and their cluster centers, termed  $D$ , in comparison to all other placements.

*Proof:* Assume that inserting object  $O$  into another cluster  $C'$  results in a smaller average distance between all objects and their cluster centers, denoted  $D'$ , than  $D$ . Since  $C'$  is not the nearest cluster of  $O$ ,  $M'(O, C) \leq M'(O, C')$ . According to Theorem 1, we have  $D \leq D'$ , which contradicts the initial assumption.  $\square$

In essence, Lemma 2 suggests how to achieve a locally optimal clustering during continuous clustering. Globally optimal clustering appears to be unrealistic for continuous clustering of moving objects—it is not realistic to frequently re-cluster all objects, and we have no knowledge of future updates.

Next, we observe that use of the Euclidean distance among objects at the time a clustering is performed or updated can be expected to be quite sub-optimal for our setting, where we are to

maintain a clustering across time. This is because the Euclidean distance only measures the difference of object positions at a single point in time, while  $M'$  measures the total difference during a time interval. It may occur frequently that objects close to each other at a point in time may be relatively far apart at later times. Therefore, even if the Euclidean distance between the object and the cluster center is at first small, the corresponding  $M'$  value could be larger, meaning that the use of the Euclidean distance results in larger average distance between objects and their cluster centers.

We proceed to consider the effect of updates during the clustering. Let  $F(t)$ , where  $0 < F(t) \leq 1$  and  $0 \leq t \leq U$ , denote the fraction of objects having their update interval being equal to  $t$ . We define the weight value  $w_x$  at time  $t_x$ , where  $0 \leq t_x \leq U$ , as follows:

$$w_x = \int_{t_x}^U F(t) dt \quad (3)$$

This weight value can reflect the update behavior. The reasons are as follows. The update interval of any object is less than the maximum update time  $U$ . After the initial cluster construction, the probability that an object will be updated before time  $t_x$  is  $\int_0^{t_x} F(t) dt$ . Because  $\int_0^U F(t) dt = 1$ , the probability that an objects will not be updated before time  $t_x$  is then  $1 - \int_0^{t_x} F(t) dt = \int_{t_x}^U F(t) dt$ . This weight value gives the 'validity' time of an object. In other words, it indicates the importance of the object's position at time  $t_x$ .

Moreover, the weight value also satisfies the property that  $t_x \leq t_y$  implies  $w_x \geq w_y$ . Let  $t_x \leq t_y$ . Then:

$$\begin{aligned} w_x - w_y &= \int_{t_x}^U F(t) dt - \int_{t_y}^U F(t) dt \\ &= \int_{t_x}^{t_y} F(t) dt + \int_{t_y}^U F(t) dt - \int_{t_y}^U F(t) dt \\ &= \int_{t_x}^{t_y} F(t) dt \geq 0 \end{aligned}$$

In the empirical study, next, we use versions of dissimilarity measure  $M$  that sum values at sample time points, rather than the boundary (integral) case considered in this section. This is done mainly for simplicity of computation.

## V. EMPIRICAL PERFORMANCE STUDIES

We proceed to present results of empirical performance studies of the proposed clustering algorithm. We first introduce the experimental settings. We then compare our proposal with the existing K-means and Birch clustering algorithms. Finally, we study the properties of our algorithm while varying several pertinent parameters.

### A. Experimental Settings

All experiments are conducted on a 2.6G Hz P4 machine with 1Gbyte of main memory. The page size is 4K bytes, which results in a node capacity of 170 objects in the MC data structures. We assign two pages to each cluster.

Due to the lack of appropriate, real moving object datasets, we use synthetic datasets of moving objects with positions in the square space of size  $1000 \times 1000$  units. We use three types of generated datasets: uniform distributed datasets, Gaussian distributed datasets, and network-based datasets. In most experiments, we use uniform data. Initial positions of all moving objects are chosen at random, as are their movement directions. Object speeds are also chosen at random, within the range of 0 to 3. In the

Gaussian datasets, the moving object positions follow a Gaussian distribution. The network-based datasets are constructed by using the data generator for the COST benchmark [5], where objects move in a network of two-way routes that connect a given number of uniformly distributed destinations. Objects start at random positions on routes and are assigned at random to one of three groups of objects with maximum speeds of 0.75, 1.5, and 3. Whenever an object reaches one of the destinations, it chooses the next target destination at random. Objects accelerate as they leave a destination, and they decelerate as they approach a destination. One may think of the space unit as being kilometers and the speed unit as being kilometers per minute. The sizes of datasets vary from 10K to 100K. The duration in-between updates to an object ranges from 1 to  $U$ , where  $U$  is the maximum update time.

Unless stated otherwise, we use the decreasing weight value as defined in equation 3, and we set the interval between sample timestamps to be 10. We store the event queue and the hash table in memory. We quantify the clustering effect by the average radius, and we examine the construction and update cost in terms of both I/Os and CPU time.

Table I offers an overview of the parameters used in the ensuing experiments. Values in bold denote default values.

TABLE I  
PARAMETERS AND THEIR SETTINGS

Parameter	Setting
Page size	4K
Node capacity	170
Cluster capacity	340
Maximum update time	60
Type of weight values	<b>Decreasing</b> , Equal
Interval between sample timestamps	5, <b>10</b> , 20, 30, 60
Dataset size	10K, ..., <b>100K</b>

### B. Comparison with Clustering Algorithms for Static Databases

For comparison purposes, we choose the K-means and Birch algorithms, which are representative clustering algorithms for static databases. To directly apply both the K-means and Birch algorithms to moving objects, both have to re-compute after every update, every  $k$  updates, or at regular time intervals in order to maintain each clustering effectiveness.

The number of clusters generated by MC is used as the desired number of clusters for the K-means and Birch algorithms. Other parameters of Birch are set similarly to those used in the literature [28]: (i) memory size is 5% of the dataset size; (ii) the initial threshold is 0.0; (iii) outlier-handling is turned off; (iv) the maximum input range of phase 3 is 1000; (v) the number of refinement passes in phase 4 is one. We then study the average radius across time. The smaller the radius, the more compact the clusters.

1) *Clustering Effect without Updates*: In this initial experiment, we evaluate the clustering effect of all algorithms across time assuming that no updates occur. Clusters are created at time 0, and the average radius is computed at each time unit. It is worth noting that the weight values in MC are equal to 1 as there are no updates. Figure 10 shows that the average cluster radius grows much faster for the K-means and Birch algorithms than for the MC algorithm, which intuitively means that MC clusters remain “valid” longer than do K-means and Birch clusters. Specifically,

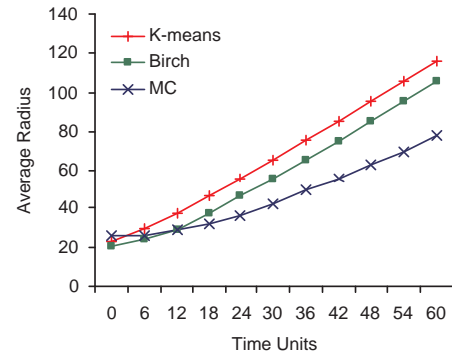


Fig. 10. Clustering Effect without Updates

at time 60, the average radii of the K-means and the Birch clusters are more than 35% larger than that of the MC clusters.

Algorithm MC achieves its higher cluster longevity by considering both object positions and velocities, and hence the moving objects in the same clusters have similar moving trend and may not expand the clusters too fast.

Observe also that the radii of the K-means and the Birch clusters are slightly smaller than those of the MC clusters during the first few time units. This is so because the MC algorithm aims to achieve a small cluster radius along the cluster’s entire life time, instead of achieving a small initial radius. For example, MC may place objects that are not very close at first, but may get closer later, in the same cluster.

2) *Clustering Effect with Updates*: In this experiment, we use the same dataset as in the previous section to compare the clusters maintained incrementally by the MC algorithm when updates occur with the clusters obtained by the K-means and the Birch algorithms, which simply recompute their clusters each time the comparison is made. Although the K-means and the Birch clusters deteriorate quickly, they are computed to be small at the time of computation and thus represent the near optimal cases for clustering.

Figure 11 shows the average radii obtained by all the algorithms as time progresses. Observe that the average radii of the MC clusters are only slightly larger than those of the K-means and the Birch clusters. Note also that after the first few time units, the average radii of the MC clusters do not deteriorate.

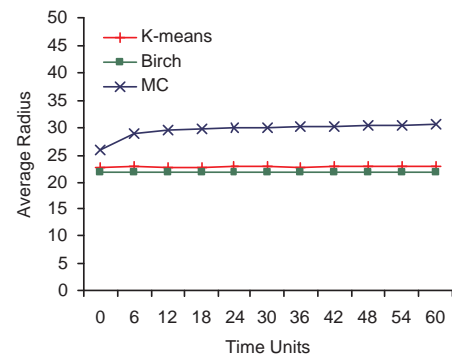


Fig. 11. Clustering Effect with Updates

3) *Clustering Effect with Dataset Size*: We also study the clustering effect when varying the number of moving objects;



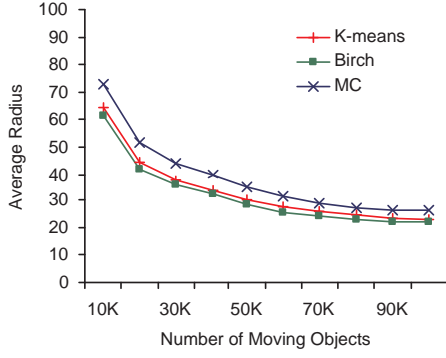


Fig. 12. Clustering Effect with Varying Number of Moving Objects

Figure 12 plots the average radius. The clustering produced by the MC algorithm is competitive for any size of dataset compared to those of the K-means and the Birch algorithms. Moreover, in all algorithms, the average radius decreases as the dataset size increases. This is because the capacity of a cluster is constant (in our case twice the size of a page), and the object density increases.

4) *Clustering Effect with Different Data Distributions:* Next, we study the clustering effect in different types of datasets. We test two network-based datasets with 100 and 500 destinations, respectively, and one Gaussian dataset. As shown in Figure 13, the average radii obtained by the MC algorithm are very close to those obtained by the K-means and Birch algorithms, especially for the network-based and Gaussian datasets. This is because objects in

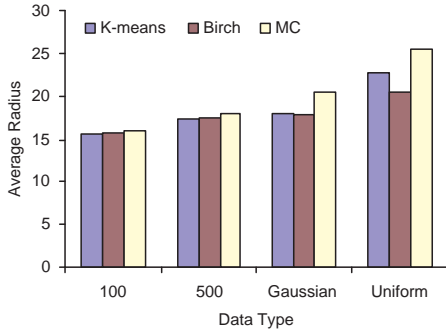


Fig. 13. Clustering Effect with Different Data Distributions

the network-based datasets move along the roads, enabling the MC algorithm to easily cluster those objects that move similarly. In the Gaussian dataset, objects concentrate in the center of the space; hence there are higher probabilities that more objects move similarly, which leads to better clustering by the MC algorithm. These results indicate that the MC algorithm is more efficient for objects moving similarly, which is often the case for vehicles moving in road networks.

5) *Inter Cluster Distance:* In addition to using the average radius as the measure of clustering effect, we also test the average inter cluster distance. The average inter cluster distance of a cluster  $C$  is defined as the average distance between the center of cluster  $C$  and the centers of all the other clusters. Generally, the larger the inter cluster distance, the better the clustering quality. Figure 14 shows the clustering results in different types

of datasets. We can observe that the average inter cluster distance of MC clusters is slightly larger than those of K-means and Birch clusters in the network-based datasets. This again demonstrates that the MC algorithm may be more suitable for moving objects such as vehicles that move in road networks.

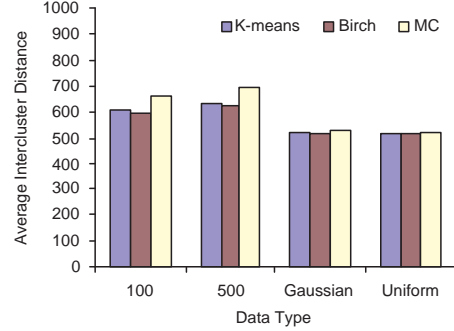


Fig. 14. Inter Cluster Distance

6) *Clustering Speed:* Having considered clustering quality, we proceed to compare the efficiency of cluster construction and maintenance for all three algorithms. Since K-means is a main-memory algorithm, we assume all the data can be loaded into main memory, so that Birch and MC also run entirely in main memory.

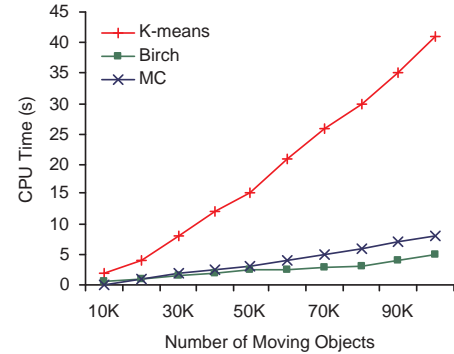


Fig. 15. Construction Time

We first construct clusters at time 0 for all algorithms. Figure 15 compares the CPU times for different dataset sizes. We observe that MC outperforms K-means, with a gap that increases with increasing dataset size. Specifically, in the experiments, MC is more than 5 times faster than K-means for the 100K dataset.

In comparison to Birch, MC is slightly slower when the dataset becomes large. The main reason is that Birch does not maintain any information between objects and clusters. This can result in time savings in Birch when MC needs to change object labels during the merging or splitting of clusters. However, construction is a one-time task, and this slight construction overhead in MC is useful because it enables efficient support for the frequent updates that occur in moving-object databases.

After the initial construction, we execute updates until the maximum update time. We apply two strategies to enable the K-means and Birch algorithms to handle updates without any modifications to the original algorithms. One is the extreme case where the dataset is re-clustered after every update, labeled “per update.”

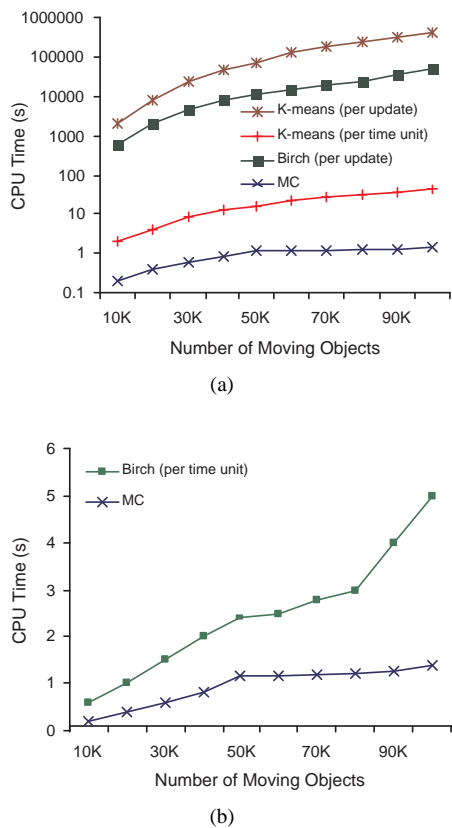


Fig. 16. Maintenance Time

The other re-clusters the dataset once every time unit, labeled “per time unit.” Figure 16 shows the average computational costs per time unit of all the algorithms for different dataset sizes. Please note that the  $y$ -axis in Figure 16(a) uses a log scale, which makes the performance gaps between our algorithms and other algorithms seem narrow. Actually, the MC algorithm achieves significant better CPU performance than both variants of each of K-means and Birch. According to Figure 16(a), the MC algorithm is up to  $10^6$  times and 50 times faster than the first and the second variant of K-means, respectively. When comparing to Birch, the MC algorithm is up to  $10^5$  times faster than the first variant of Birch (Figure 16(a)) and up to 5 times faster than the second variant of Birch (Figure 16(b)).

These findings highlight the adaptiveness of the MC algorithm. The first variants recompute clusters most frequently and thus have by far the worst performance. The second variants have lower recomputation frequency, but are then as a result not able to reflect the effect of every update in its clustering (e.g., they are unable to support a mixed update and query workload). In contrast, the MC algorithm does not do re-clustering, but instead incrementally adjusts its existing clusters at each update.

### C. Properties of the MC Algorithm

We proceed to explore the properties of the MC algorithm, including its stability and performance under various parameters and its update I/O cost.

1) *The Number of Clusters*: We first study the number of clusters, varying the dataset size and time. As shown in Figure 17, the number of clusters remains almost constant for the same

dataset as time passes. Recall also Figure 11 (in Section V-B.2). We can derive from these results that the MC algorithm maintains a similar number of clusters and similar sizes of radii for the same dataset as time passes, which indicates that the MC algorithm has stable performance. In other words, the passing of time has almost no effect on the results produced by the MC algorithm.

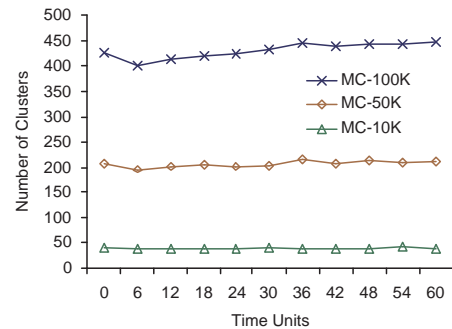


Fig. 17. Number of Clusters with Different Data Sizes

2) *Effect of Weight Values*: Next, we are interested in the behavior of the MC algorithm under the different types of weight values used in the dissimilarity measurements. We take two types of weight values into account: (i) decreasing weights (see equation 3):  $w_j > w_{j+1}$ ,  $1 \leq j \leq k-1$ ; (ii) equal weights:  $w_j = w_{j+1}$ ,  $1 \leq j \leq k-1$ .

From now on, we use the total radius (i.e., the product of the average radius and the number of clusters) as the clustering effect measurement since the numbers of clusters are different for the different weight values. Figure 18 shows the total radius of clusters generated by using these two types of weight values. It is not surprising that the one using decreasing weight values

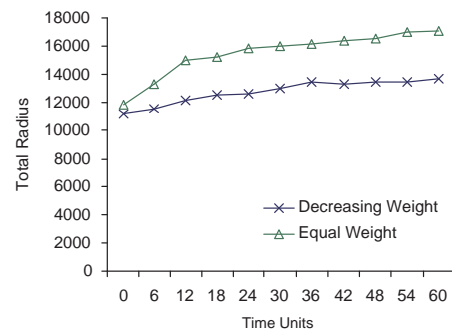
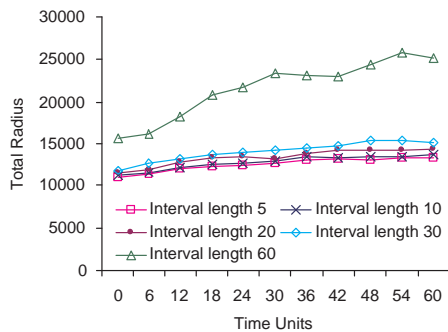


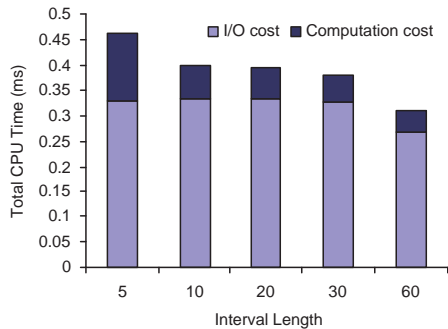
Fig. 18. Clustering Effect with Different Types of Weight Values

yields better performance. As we mentioned before, the closer to the current time, the more important the positions of moving objects are because later positions have higher probabilities of being changed by updates.

3) *Effect of Time Interval Length between Sample Points*: Another parameter of the dissimilarity measurement is the time interval length between two consecutive sample positions. We vary the interval length and examine the performance of the MC algorithm as time progresses (see Figure 19(a)). As expected, we can see that the one with the shortest interval length has the smallest radius (i.e., best clustering effect). However, this does not mean that the shortest interval length is an optimal value



(a) Clustering Effect



(b) Maintenance Time

Fig. 19. Effect of Different Interval Lengths

considering the overall performance of the MC algorithm. We need to consider the time efficiency with respect to the interval length. In addition, we also observe that the difference between the time intervals equal to 60 and 30 is much wider than the others. The possible reason is that when time interval is 60, there are only two sample points (start and end points of an object trajectory), which are not able to differentiate the two situations shown in Figure 4. Therefore, it is suggested to use no less than three sample points so that the middle point of a trajectory can be captured.

Figure 19(b) shows the maintenance cost of the MC algorithm when varying the time interval length. Observe that the CPU time decreases with the increase of the time interval length, while the I/O cost (expressed in milliseconds) does not change much. This is because the longer time interval results in less sample positions, and hence less computation. In contrast, the I/O cost is mainly due to the split and merge events. When the time interval length increases, the dissimilarity measurement tends to be less tight, which results in less split and merge events.

Considering the clustering effect and time efficiency together, the time interval length should not be larger than 30, as we need at least three sample points, and it should not be too small, as this will yield unnecessarily many computations. Therefore, we choose the number of sample points to be a little more than 3 as a tradeoff. In our experiments, the number of sample points is 6, corresponding to the time interval length 10.

4) *Update I/O Cost*: We now study the update I/O cost of the MC algorithm solely. We vary the dataset size from 10K to 100K and run the MC algorithm for the maximum update interval. Figure 20 records the average update cost. As we can see, the update cost is only 2 to 5 I/Os because each insertion or deletion

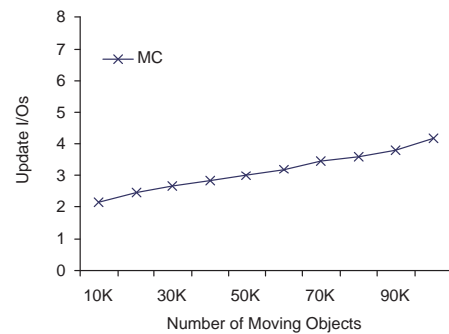


Fig. 20. Update I/O Cost

usually affects only one or two clusters. This suggests that the MC algorithm has very good update performance.

## VI. CONCLUSION

This paper proposes a fast and effective scheme for the continuous clustering of moving objects. We define a new and general notion of object dissimilarity, which is capable of taking future object movement and expected update frequency into account, with resulting improvements in clustering quality and running time performance. Next, we propose a dynamic summary data structure for clusters that is shown to enable frequent updates to the data without the need for global re-clustering. An average radius function is used that automatically detects cluster split events, which, in comparison to existing approaches, eliminates the need to maintain bounding boxes of clusters with large amounts of associated violation events. In future work, we aim to apply the clustering scheme in new applications.

## ACKNOWLEDGEMENT

The work of Dan Lin and Beng Chin Ooi was in part funded by an A\*STAR project on spatial-temporal databases.

## REFERENCES

- [1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining application. In *Proc. ACM SIGMOD*, pp. 94–105, 1998.
- [2] M. Ankerst, M. Breunig, H.P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proc. ACM SIGMOD*, pp. 49–60, 1999.
- [3] Applied Generics. RoDIN24. [www.appliedgenerics.com/downloads/RoDIN24-Brochure.pdf](http://www.appliedgenerics.com/downloads/RoDIN24-Brochure.pdf), 2006.
- [4] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Algorithms*, 31(1): 1–28, 1999.
- [5] C. S. Jensen, D. Tiesyte, and N. Tradisaukas. The COST Benchmark-Comparison and Evaluation of Spatio-temporal Indexes. *Proc. DASFAA*, pp. 125–140, 2006.
- [6] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38: 293–306, 1985.
- [7] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proc. ACM SIGMOD*, pp. 73–84, 1998.
- [8] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-line discovery of dense areas in spatio-temporal databases. In *Proc. SSTD*, pp. 306–324, 2003.
- [9] S. Har-Peled. Clustering motion. *Discrete and Computational Geometry*, 31(4): 545–565, 2003.
- [10] V. S. Iyengar. On detecting space-time clusters. In *Proc. KDD*, pp. 587–592, 2004.
- [11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

- [12] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient  $B^+$ -tree based indexing of moving objects. In *Proc. VLDB*, pp. 768–779, 2004.
- [13] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *Proc. SSTD*, pp. 364–381, 2005.
- [14] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering algorithm using dynamic modeling. In *IEEE Computer*, 32(8):68–75, 1999.
- [15] D. Kwon, S. Lee, and S. Lee. Indexing the current positions of moving objects using the lazy update R-tree. In *Proc. MDM*, pp. 113–120, 2002.
- [16] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proc. KDD*, pp. 617–622, 2004.
- [17] J. Macqueen. Some methods for classification and analysis of multivariate observations. In *Proc. Berkeley Symp. Math. Statist.*, pp. 281–297, 1967.
- [18] S. Nassar, J. Sander, and C. Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. In *Proc. ACM SIGMOD*, pp. 467–478, 2004.
- [19] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *Proc. VLDB*, pp. 144–155, 1994.
- [20] J. M. Patel, Y. Chen, and V. P. Chakka. STRIPES: An efficient index for predicted trajectories. In *Proc. ACM SIGMOD*, pp. 637–646, 2004.
- [21] S. Saltinis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD*, pp. 331–342, 2000.
- [22] M. Spiliopoulou, I. Ntoutsis, Y. Theodoridis, and R. Schult. MONIC: modeling and monitoring cluster transitions. In *Proc. KDD*, pp. 706–711, 2006.
- [23] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. ACM SIGMOD*, pp. 611–622, 2004.
- [24] Y. Tao, D. Papadias, and J. Sun. The TPR\*-tree: An optimized spatio-temporal access method for predictive queries. In *Proc. VLDB*, pp. 790–801, 2003.
- [25] W. Wang, J. Yang, and R. Muntz. Sting: a statistical information grid approach to spatial data mining. In *Proc. VLDB*, pp. 186–195, 1997.
- [26] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *Proc. ACM SIGMOD*, pp. 443–454, 2004.
- [27] Q. Zhang and X. Lin. Clustering moving objects for spatio-temporal selectivity estimation. In *Proc. ADC*, pp. 123–130, 2004.
- [28] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD*, pp. 103–114, 1996.



**Dan Lin** received the B.S. degree (First Class Honors) in Computer Science from Fudan University, China in 2002, and the Ph.D. degree in Computer Science from the National University of Singapore in 2007. Currently, She is a visiting scholar in the Department of Computer Science at Purdue University, USA. Her main research interests cover many areas in the fields of database systems and information security. Her current research includes geographical information systems, spatial-temporal databases, location privacy, and access control policies.



**Beng Chin Ooi** received the B.S. (First Class Honors) and Ph.D. degrees from Monash University, Australia in 1985 and 1989 respectively. He is currently a professor of computer science at the School of Computing, National University of Singapore. His current research interests include database performance issues, index techniques, XML, spatial databases and P2P/grid Computing. He has published more than 100 conference/journal papers and served as a PC member for a number of international conferences (including SIGMOD, VLDB, ICDE, EDBT and DASFAA). He is an editor of *GeoInformatica*, the *Journal of GIS*, *ACM SIGMOD Disc*, *VLDB Journal* and the *IEEE TKDE*. He is a member of the ACM and the IEEE.



**Christian S. Jensen** (Ph.D., Dr.Techn.) is a Professor of Computer Science at Aalborg University, Denmark, and an Adjunct Professor at Agder University College, Norway.

His research concerns data management and spans issues of semantics, modeling, and performance. With his colleagues, he has published widely on these subjects. With his colleagues, he receives substantial national and international funding for his research.

He is a member of the Danish Academy of Technical Sciences, the EDBT Endowment, and the VLDB Endowment's Board of Trustees. He received Ib Henriksen's Research Award 2001 for his research in mainly temporal data management and Telenor's Nordic Research Award 2002 for his research in mobile services.

His service record includes the editorial boards of ACM TODS, IEEE TKDE and the IEEE Data Engineering Bulletin. He was the general chair of the 1995 International Workshop on Temporal Databases and a vice PC chair for ICDE 1998. He was PC chair or co-chair for the Workshop on Spatio-Temporal Database Management, held with VLDB 1999, for SSTD 2001, EDBT 2002, VLDB 2005, MobiDE 2006, and MDM 2007. He is a vice PC chair for ICDE 2008. He has served on more than 100 program committees.

He serves on the boards of directors and advisors for a small number of companies, and he serves regularly as a consultant.