Clustering Moving Objects *

Yifan Li
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
USA
yifanli@cs.uiuc.edu

Jiawei Han
Department of Computer Science
University of Illinois
Urbana-Champaign, IL 61801
USA
hanj@cs.uiuc.edu

Jiong Yang
EECS Department
Case Western Reserve University
Cleveland, OH 44106
USA
jiong@eecs.cwru.edu

ABSTRACT

Due to the advances in positioning technologies, the real time information of moving objects becomes increasingly available, which has posed new challenges to the database research. As a long-standing technique to identify overall distribution patterns in data, clustering has achieved brilliant successes in analyzing static datasets. In this paper, we study the problem of clustering moving objects, which could catch interesting pattern changes during the motion process and provide better insight into the essence of the mobile data points. In order to catch the spatial-temporal regularities of moving objects and handle large amounts of data, micro-clustering [20] is employed. Efficient techniques are proposed to keep the moving micro-clusters geographically small. Important events such as the collisions among moving micro-clusters are also identified. In this way, high quality moving micro-clusters are dynamically maintained, which leads to fast and competitive clustering result at any given time instance. We validate our approaches with a through experimental evaluation, where orders of magnitude improvement on running time is observed over normal K-Means clustering method [14].

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications - Data Mining

General Terms: Algorithms.

Keywords: Moving object, micro-cluster, clustering, algorithms.

1. INTRODUCTION

The extensive existence of moving objects and the highly developed capability of location awareness boost the requests of a diverse range of services that involve the exploit of knowledge of the changing positions of objects. This has posed new challenges to database research, including data storage, data analysis, query processing and result presentation. Not surprisingly, the research on moving object has attracted increasing attention in the community [10, 16, 17, 19] in recent years.

Clustering analysis, which groups similar data to reveal overall distribution patterns and interesting correlations in datasets, has a number of applications in data compression, image processing, pattern recognition, and market research. Consequently, as a central problem in computer science, it has been an active research area for a long time. Many successful and scalable approaches have been proposed [1, 2, 8, 12, 14, 15, 18, 20], which have achieved salient successes on analyzing static datasets.

In spite of the extensive study of clustering problem and the escalating popularity of research on moving objects, there is not much work conducted on clustering analysis on mobile data. In this paper, we study the problem of clustering moving objects, which is able to unveil some interesting and important patterns that could be invisible with clustering on static data. Clustering analysis on moving objects has numerous applications, including weather forecasting (e.g., cyclone clustering [5]), traffic jam prediction, animal migration analysis, mobile computing, and outlier analysis.

The difficulty of the problem results from the innate characteristic of moving objects. The continuous location changes of large quantities of data points make the whole picture chaotic, which renders it non-trivial to capture the trends of data. Furthermore, as shown in [9], even minor location changes of data points could lead to significantly different clustering result.

One straightforward way to handle the problem is to cluster the dataset periodically. However, if the interval between two consecutive clusterings is short, this approach would be quite expensive when the number of objects is large, since it does not take advantage of the information gathered from previous clusterings. In addition, such brute force method fails to identify the nature of moving objects (e.g., some groups of data moving together) and is blind to trends of motion.

In order to handle very large datasets, which are not uncommon in reality, *micro-clustering* was originally exploited by Zhang et al. in [20]. The concept *micro-cluster* proposed there indicates a group of data that are so close to each other

*The work was supported in part by U.S. National Science Foundation NSF IIS-03-08215, University of Illinois, and an IBM Faculty Award. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

that they are likely to belong to one cluster. In this paper, we extend the concept to *moving micro-cluster*, which denotes a group of objects that are not only close to each other at current time, but also likely to move together for a while. In principle, those moving micro-clusters reflect some closely moving objects, based on which the high quality clustering result can be obtained naturally.

Due to the difference of positions and velocities of objects within a moving micro-cluster, the objects tend to *scatter* after a period of time, which may break the requirement of micro-cluster since they will not be adjacent to each other any more. To avoid the deterioration of the quality of moving micro-clusters, those micro-clusters may split and get reorganized at appropriate time instances. Thus the micro-clusters are kept geographically *compact* at any time, which lays a solid foundation for successive clusterings. Efficient algorithms are proposed to catch those time instances and perform the organizations.

As frequently observed in the experiments, moving micro-clusters may *bump* into each other. The knowledge of when those events occur can certainly provide direct clues for clusterings taking places around those time instances. The events are also identified and managed in the paper. In view of the kinetic property of our data structures, the algorithm can accommodate the updates of moving objects in a real time fashion. Typically it is desirable to provide multi-level data analysis for prohibitively large datasets. The functionality is naturally supported based on the following observation. A moving micro-cluster as a whole could be viewed as a moving object, just like that micro-clusters are safely treated as unit components of clustering in [20].

The rest of the paper is organized as follows. Section 2 introduces some background knowledge. Section 3 provides the description of moving micro-clusters. We present our algorithm and the experiments in Section 4 and Section 5 respectively. Related work is introduced in Section 6. Finally, we conclude the paper in Section 7.

2. BACKGROUND

In this paper, we employ the model used in [16, 17], where the objects are assumed to move in a piecewise linear manner. Namely, an object moves along a straight line with some constant speed till it changes the direction and/or speed. If an object deviates significantly from the expected position due to some reason (e.g., the variation of its velocity), the object is responsible for reporting the new velocity.

We consider the objects in the R^n space with some $n > 0$. In addition, we assume that time is an additional continuous dimension that can be represented as R. The location of an object is denoted by a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$. It is a function of time t and can be written as $\vec{x}(t) = \vec{x}(t_0) + \vec{v}t$, where $\vec{x}(t_0)$ is the initial location of the object at some referential instance t_0, and $\vec{v} \in R^n$ is the velocity vector. For the remainder of this paper, we mainly consider the case of $n = 2$, which is ready to be extended to higher dimensions.

In the 2-D space, each moving object o is represented by a 5-tuple $(x_o, y_o, vx_o, vy_o, t_o)$, which indicates that at time instance t_o, the object o is at location (x_o, y_o) and with velocity (vx_o, vy_o). It is also called the $profile$ of moving object o, because it uniquely determines the track of o. The subscript may be omitted if no confusion is possible. Euclidean distance is used in the paper for simplicity. In principle, any other eligible metric will work with the algorithm. We also use (x_o^t, y_o^t) / (vx_o^t, vy_o^t) to denote the location / velocity of object o at time t. In the paper, "object" and "(data) point" are used interchangeably.

3. MOVING MICRO-CLUSTERS

Property 1. The profile (x, y, vx, vy, t) of a moving object o can be equivalently written as $(x + (t' - t)vx, y + (t' - t)vy, vx, vy, t')$, if the velocity remains during time interval $(t, t']$ $(t < t')$.

As stated in Section 1, a moving micro-cluster is composed of moving objects that are expected to stay together for a while during the motion. As defined below, it has its own profile which reflects the moving characteristic of the whole group of objects.

Definition 1. A moving micro-cluster MMC is composed of n similar[1] moving objects $(x_i, y_i, vx_i, vy_i, t)$[2], where $i = 1, 2, ..., n$. The profile of MMC is (x, y, vx, vy, t), where $(x, y, vx, vy) = (\sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n vx_i, \sum_{i=1}^n vy_i)/n$. We call (x_{MMC}^t, y_{MMC}^t) / (vx_{MMC}^t, vy_{MMC}^t) the center / velocity of MMC at time t.

The following property shows that the profile of a moving micro-cluster indeed characterizes the motion of the group of the objects.

Property 2. Given a MMC (x, y, vx, vy, t), the center of the moving micro-cluster will become $((x + (t' - t)vx, y + (t' - t)vy)$ at time t', if there is no membership update of MMC and the velocities of the objects remain during time interval $(t, t']$ $(t < t')$.

The property directly follows Property 1 and Definition 1. Notice that although Property 2 holds regardless of the distributions of the objects' locations and velocities, the moving micro-cluster can catch the moving feature of the objects only when it consists of objects with similar profiles.

Being made up of similar objects, the moving micro-cluster serves as a good summary of its cluster members, as the movement of the micro-cluster gives an insightful overview of the whole motion process. Furthermore, if we view the center of a micro-cluster as its location, a moving micro-cluster can be treated as a normal moving object. Accordingly, a micro-cluster could contain moving objects and other micro-clusters, which enables us to build a hierarchical structure. On the other hand, a moving object is in fact a single member moving micro-cluster. Therefore, in the remaining parts of the paper, we do not make distinction between a moving object and a moving micro-cluster, unless otherwise stated.

As we can imagine, the members of a moving micro-cluster should be updated with time so as to keep the micro-cluster in *high quality* (i.e., the micro-cluster always contains similar data points). The addition / deletion of components occurring to a moving micro-cluster may lead to the update of its profile. In order to dynamically maintain the

[1]The similarity is measured by distance on profiles of objects, which is elaborated in Section 4.1. Basically, similar objects are expected to have similar initial locations and similar velocities.

[2]The referential time instances of the objects can be unified by Property 1.

profile, we define the clustering feature as summary information of a micro-cluster, which was first proposed in [20]. Specifically, we define *clustering feature* (CF) of the moving micro-cluster (x, y, vx, vy, t) as (SX, SY, SVX, SVY, N, t), where N is the number of moving objects that belong to the micro-cluster, $SX = \sum_{i=1}^{N} x_i$, $SY = \sum_{i=1}^{N} y_i$, and so on. Obviously $(x, y, vx, vy)^T = (SX, SY, SVX, SVY)^T/N$.

The clustering feature can be updated online according to the following lemmas.

LEMMA 1. *Assume MMC_1 with $CF1 = (SX_1, SY_1, SVX_1, SVY_1, N_1, t_1)$ joins a disjoint MMC_2 with $CF_2 = (SX_2, SY_2, SVX_2, SVY_2, N_2, t_2)$ at time t $(t_1, t_2 < t)$, then the CF_2 becomes $(SX, SY, SVX_1 + SVX_2, SVY_1 + SVY_2, N_1 + N_2, t)$, where $SX = SX_1 + (t - t_1)SVX_1 + SX_2 + (t - t_2)SVX_2$ and $SY = SY_1 + (t - t_1)SVY_1 + SY_2 + (t - t_2)SVY_2$.*

LEMMA 2. *Assume MMC_1 with $CF1 = (SX_1, SY_1, SVX_1, SVY_1, N_1, t_1)$ leaves MMC_2 with $CF_2 = (SX_2, SY_2, SVX_2, SVY_2, N_2, t_2)$ at time t $(t_1, t_2 < t)$, (MMC_2 contains MMC_1 before t), then CF_2 becomes $(SX, SY, SVX_2 - SVX_1, SVY_2 - SVY_1, N_2 - N_1, t)$, where $SX = SX_2 + (t - t_2)SVX_2 - SX_1 - (t - t_1)SVX_1$ and $SY = SY_2 + (t - t_2)SVY_2 - SY_1 - (t - t_1)SVY_1$.*

The proof involves the above definitions, properties and some straightforward algebra, which is omitted here due to the space limitation. In consequence, the profiles of moving micro-clusters can be dynamically maintained without repeated scans over the components upon updates.

4. MOVING MICRO-CLUSTERING (MMC) ALGORITHM

Since moving micro-clusters are aimed at capturing some closely moving objects, the initialization of such micro-clusters requires the consideration of the velocity information as well as the initial location information. The initial construction of moving micro-clusters is introduced in Section 4.1.

In general, though we take both location and velocity information into consideration when performing the initial builds of the moving micro-clusters, the objects within one micro-cluster still tend to spread after some time, due to the different velocities and initial locations. Namely, after that time instance the objects are not close to each other any longer, which invalidates the micro-cluster by definition. To catch such time instances, we tightly bound each micro-cluster by a rectangle, whose size grows with time. When the size of the bounding rectangle reaches some threshold, it is high time that a split should be carried out to keep the micro-cluster compact. To minimize the cost of split, we choose to remove some data points on the boundaries of the rectangle. Some heuristics for selecting the *victim* (the point to be removed) are suggested to delay the next split as much as possible. The victim may join other micro-clusters or become a stand-alone micro-cluster. We call such phenomena a *split* event. (Abusing the terminology slightly, "event" is also used to represent the time instance of the event.) The identification and management of split events are presented in detail in Section 4.2.

On the other hand, we often observe that the moving micro-clusters *bump* into each other, when we say *collision* events occur. The identification of such events, which is described in Section 4.3, would provide very useful knowledge about the clusterings that are requested around the time instances, since adjacent micro-clusters should be included in one cluster.

A priority queue Q that is keyed by time is constructed to store those split/collision events. Under this schema, high quality micro-clusters and useful information are kept, which are ready to be fed to the generic algorithm to ensure the efficient and effective micro-clustering.

Finally, we give out the time/space complexity analysis of the algorithm in Section 4.4.

4.1 Moving Micro-cluster Initialization

We select objects with similar profiles to form micro-clusters by invoking a generic clustering algorithm (K-Means algorithm is used in our experiments) with the distance metric considering both location and velocity information. Specifically, $dist^2(o_1, o_2) = (x_{o_1} - x_{o_2})^2 + (y_{o_1} - y_{o_2})^2 + (\alpha(vx_{o_1} - vx_{o_2}))^2 + (\alpha(vy_{o_1} - vy_{o_2}))^2$, where α $(\alpha > 1)$ is a weight associated with the velocity attributes since it plays a more important role (than the initial locations) in determining the spatial distances between o_1 and o_2 in the future.

4.2 Split Events

4.2.1 Bounding Rectangles

We use the bounding rectangle of a micro-cluster to measure how compact it is. The *bounding rectangle* of a moving micro-cluster MMC at time t is the minimum orthogonal rectangle (whose edges are aligned with the axes) that contains all the components. It is denoted by $((x^t_{min}, y^t_{min}), (x^t_{max}, y^t_{max}))$, where $x^t_{min} = \min_{i \in MMC} x^t_i$, $y^t_{min} = \min_{i \in MMC} y^t_i$, and so on. The superscript is omitted if clear from the context. Those points on the boundary of the bounding rectangle are called *extreme points*.

4.2.2 Identifying the Split Events

Typically the width and height of the bounding rectangle increase with time. A *split* event occurs when the width or height reaches some pre-defined threshold L. Note that L depends on the size of current bounding rectangle (e.g., a split event occurs when the width/height of the bounding rectangle is 15% larger than the original value). Thus different micro-clusters may have different thresholds, since they are typically of different sizes. It is non-trivial to catch the time instance when a split event takes place, since the extreme points could change with time even when constant velocities of data points are assumed. Accordingly, the length of width (height) of the bounding rectangle is not necessarily a linear function of time, but a piece-wise linear function with the interruption points being the time instances when the extreme objects change, as illustrated in Figure 1.

Example 1. Suppose there are four 1-D moving objects: a, b, c, and d. At time 0, they are at locations 0, 10, 25, 30, with the velocity 0, 3, 2, 1, respectively. The width of the bounding segment is a piece-wise linear function as shown in Figure 1, where the right-side extreme points are d, c, b during time interval $[0, 5)$, $[5, 15)$ and $[15, +\infty)$, respectively. The threshold L and the split event t' are also indicated.

W.L.O.G., below we concentrates on the projections of moving objects on the x-axis. Now each moving object is denoted by a 3-tuple (x, vx, t). Assume that the current

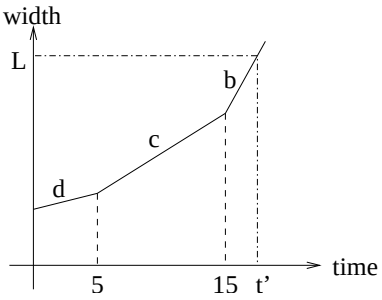


Figure 1: The width of the bounding segment is a piece-wise linear function of time.

time is t and $x^t_{max} - x^t_{min} < L$, we seek to find the time instance t' ($t' > t$) s.t. $x^{t'}_{max} - x^{t'}_{min} = L$.

It is evident that if we maintain the leftmost and rightmost points dynamically, it would be trivial to compute the split event t'. However, given a moving micro-cluster that contains n data points, the extreme points could change up to $O(n)$ times during the motion, and there could be data point joining / leaving the micro-cluster at any time, which makes the maintenance of extreme points a difficult task. W.L.O.G, we show how to keep the online information of rightmost point as follows.

We call the rightmost object change a *critical incident*. (For convenience, incident is also used to indicate the time instance when the incident occurs.) Those critical incidents can be found in the following way. With a single scan of the components, we would be able to find the nearest future time instance when the current rightmost point will be overpassed, which is exactly the next critical incident. We then continue the procedure to obtain subsequent critical events. The time complexity is $O(n^2)$ since each round of computation costs $O(n)$ and there could be $O(n)$ critical incidents during the motion.

Alternatively, we can dynamically maintain the order of all the points according to their locations. We schedule an incident that is to be stored in a priority queue pq when the two currently consecutive data points (called neighbors) meet, which leads to the exchange of their positions in the sorted list. When such incident takes places, up to two old incidents will be deleted from pq and up to two new incidents will be inserted into pq. In this way, we always have the up-to-date sorted list of the data points, which is able to tell which one is the rightmost point at any given time. Unfortunately, the number of incidents is still $O(n^2)$ in the worst case because it is easy to find cases where such neighbor changes could occur $O(n^2)$ times.

The key observation is that we do not have to maintain the total order of the data points. Instead, a partial order will be enough to indicate the rightmost point. For instance, in Example 1, no attention needs to be paid when point b and d exchange their orders at time $t = 10$, as long as we know that they are all on the left of point c. As shown in Figure 2, *kinetic heap* [3] is chosen to maintain the rightmost point dynamically.

A kinetic heap is a heap with the nodes representing the data points and the edges showing the partial order relationship (i.e., the children are on the left of their father). Thus the root is always the rightmost point. For each edge of the heap, we associate an incident stored in the priority queue pq that indicates the time instance when the child

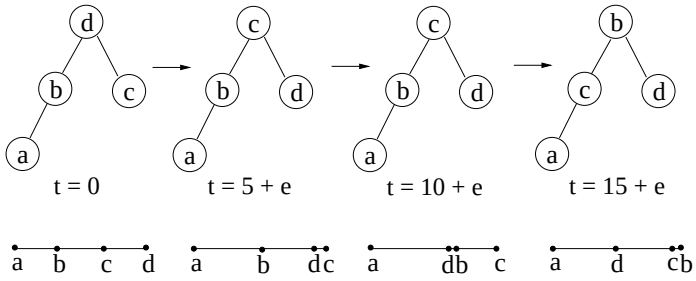


Figure 2: The kinetic heap of Example 1.

passes its father. To maintain the heap property, we process incidents sequentially from pq (which is keyed by time). For each incident, we switch the nodes connected by the edge that is associated with the incident, which will result in constant number (up to 4) modifications to be made to pq. The modifications involves deletion of old incidents and addition of new ones. The kinetic heaps of different time instances are shown in Figure 2, from which we can see that there are incidents occurring at time $t = 5$ and $t = 15$. Note that there is no incident scheduled at time $t = 10$ although point b catches up to point d at that time, because there is no edge between b and d. As a result, the heap is left intact at $t = 10$. Compared to the previous method, the maintenance of the partial order saves us unnecessary incidents occurring between internal points which cannot possibly cause the change of the extreme point.

In [6], Fonseca et al. show that there are $O(n \log n)$ incidents if there is no insertion or deletion of data points during the motion. If there is, as is our case, there are $O(n \log^2 n)$ incidents, which is proved in [4]. It is a significant improvement over the previous $O(n^2)$ complexity. An extra $O(\log n)$ factor (the cost of operations on pq) may need to be added to the running time, since the size of the priority queue is $O(n)$.

4.2.3 Handling Split Events

When a split event is observed, one of the extreme points needs to be chosen as the *victim* that will be removed to keep the bounding rectangle from getting too large. For example, assume that micro-cluster MMC has two extreme points o_1 and o_2 in the x-dimension (the leftmost one and the rightmost one, respectively). The following heuristics are used. If they are moving in different directions, we keep the one that is moving in the same direction as the micro-cluster and choose the other as the victim. It can be done by observing the signs of vx_{o_1}, vx_{o_2} and vx_{MMC}. Otherwise, we choose the one that has the larger difference from vx_{MMC}. In general, the outlier is selected as the victim in this way.

The victim could join the *nearest* micro-cluster from it, in terms of the measure used in Section 4.1. Note that this may increase the size of the bounding rectangle of the moving micro-cluster that admits the victim. Otherwise, if the nearest distance is still large, which implies that the victim is quite different from all the others, it will be treated as a new stand-alone moving micro-cluster.

4.3 Collision Events

As stated before, it is useful to keep the knowledge about the collisions among moving micro-clusters. If two points meet, we allow them to pass each other without interaction.

Because of this, two micro-clusters will *pass* each other after the collision. The observation is that during the collision, the micro-clusters should be clustered together since they are close to each other. Hence we record the time instances as the *collision events* when each micro-cluster enters/leaves the collision area with other micro-clusters. Given the profiles of micro-clusters MMC_1 and MMC_2, the events can be straightforwardly computed from the inequality $dist(MMC_1, MMC_2) < D$, where D is the threshold to define the range of collision area.

4.4 Analysis

Assume there are N moving objects and M micro-clusters $MMC_1, MMC_2, ..., MMC_M$ that contain $N_1, N_2, ..., N_M$ moving objects respectively. Usually $M \ll N$ and $\sum N_i = N$. Let U be the number of object updates, SE be the number of split events, and CE be the number of collision events. We only consider the cost of maintaining the moving micro-clusters, as the final clustering cost is dependent on the chosen clustering algorithm.

For simplicity of analysis, we assume no hierarchical structure is used. The priority queue Q that is used to store the events and updates is of size $O(SE + CE + U)$. The split events take $M \log(SE + CE + U) \sum_{SE} N_i$ time, which is $O(M \cdot SE \cdot \log N \log(SE + CE + U))$. Each collision event costs constant time. For the update handling, it costs $O(U)$ to modify the profiles and $\sum_U \log N_i = O(U \log N)$ to alter the kinetic structure, without consideration of the operation cost of Q. Besides, the maintenance of kinetic structure would take $\sum N_i \log^3 N_i = O(N \log^3 N)$ time. (If there is no object updates, the cost will be reduced to $O(N \log^2 N)$.) As a consequence, the time complexity is $O((SE \cdot M \cdot \log N + CE + U \log N) \log(SE + CE + U) + N \log^3 N)$. Considering that $SE = O(N)$, $CE = O(M^2)$, and $M \ll N$, the complexity is $O((N + U) \log(N + U) \log N + N \log^3 N)$.

The space complexity is $O(N)$, which follows the fact that the sizes of kinetic heaps add up to $O(N)$. If each micro-cluster is reasonably stable (e.g., the bird clusters formed during the migration), we do not need to keep the mapping between the micro-clusters and moving objects, which greatly reduces the space requirement to $O(M)$.

5. EXPERIMENTAL EVALUATION

In this section, we provide the empirical evaluation of our proposed algorithm. The implementation is done on a Pentium 4 PC with 512M memory under Linux 2.4.20-8, with LEDA [13] being used in support of the visualization.

Our experiments are conducted in a 2-D world with size 600x600. We have used a collection of synthetic datasets generated by our data generator. Assume time starts from 0. At time $t = 0$, we create K clusters whose centers are randomly distributed in the world. For each cluster, the number of points are uniformly distributed on (N_l, N_u). The locations and velocities of points satisfy the Gaussian distribution. For each cluster, (C_x, C_y) is the expectation of the locations of the points within it, where both of them obey the uniform distribution on $(0, 600)$. (C_{vx}, C_{vy}) is the expectation of the velocities of all the data points, where both of they are uniformly distributed on $(-3, 3)$. $\mu_{location}$, uniformly distributed on $(0, 5)$, is the standard deviation of the locations of points, and $\mu_{velocity}$, uniformly distributed on $(0, 1)$, is the standard deviation of the velocities of points. N_{mmc} denotes the number of moving micro-clusters at time 0. We control the number of points N by adjusting the values of K, N_l, N_u.

In our experiments, we choose K-Means algorithm [14] as the generic algorithm used in the micro-clustering. Give a certain time instance, we then compare the running time of the K-Means clustering on the moving objects and on the moving micro-clusters provided by our algorithm, where the former is denoted by NC (Normal Clustering) and the latter is named MMC (Moving Micro-Clustering). The *squared error function* $E = \sum_{i=1}^{k} \sum_{p \in C_i} d^2(p, c_i)$ [7] is used to measure the performance of clustering, where k is the number of clusters and c_i is the center of cluster C_i. The y-axes in Figure 3 (b) and Figure 4 (b) show the measure $1 - \frac{E(MMC) - E(NC)}{E(NC)}$, which indicate the performance of MMC compared with NC[3].

As we expected, our algorithm shows significant speedup over the normal clustering method — orders of magnitude running time improvement is observed, while slight performance loss is suffered at the same time.

First, we study the scalability with respect to the number of objects. We set the number of clusters to 100 and the number of micro-clusters to 1000. As shown in Figure 3 (a), larger difference is seen with the increment of number of data points. Sometimes our MMC algorithm is more than 100 times faster (N=100000). It is within our expectation since the running time of NC increases with the number of points, while that of MMC mainly depends on the number of micro-clusters, which is fixed here. Due to the efforts to make the bounding rectangles compact and the obtained knowledge about adjacent micro-clusters, only marginal performance loss is observed in Figure 3 (b) (e.g., less than 10% for 80000 objects), which is acceptable considering the huge improvement in running time.

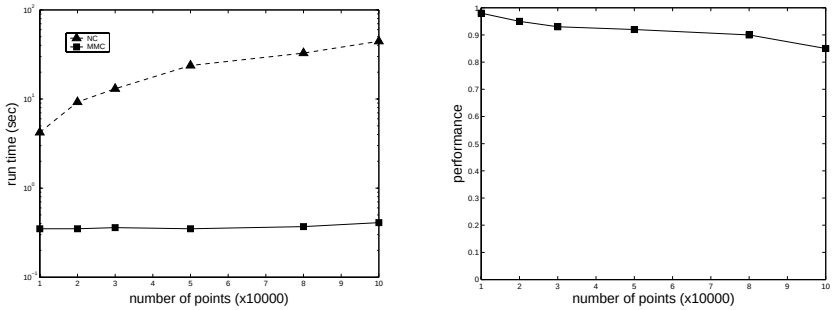
We vary the number of micro-clusters and the effect is shown in Figure 4. As opposed to Figure 3 (a), MMC algorithm shows sub-linear increment of running time with increasing number of micro-clusters, while there is no obvious running time variation of NC algorithm, since the number of data points is constant in this case (N=50000). However, our algorithm still runs orders of magnitude faster. With the increment of micro-clusters the performance is improved (Figure 4 (b)), due to the resultant finer level of granularity.

6. RELATED WORK

The research on moving object database has received increasing attention in the community. A number of data storage paradigms and query processing algorithms have been proposed recently [10, 16, 17, 19]. Due to its very extensive applications, clustering problem has been an active research topic for several decades. Various efficient and scalable clustering algorithms have been proposed [1, 2, 8, 12, 14, 15, 18, 20]. For an elaborate survey, readers are referred to [11]. In [20], the concept of *micro-cluster* was proposed, which denotes a group of data points that are close to each other and are suitable to be treated as one unit. Our approach extends the concept, where a much more complex mechanism is required to maintain the high quality of the moving micro-clusters.

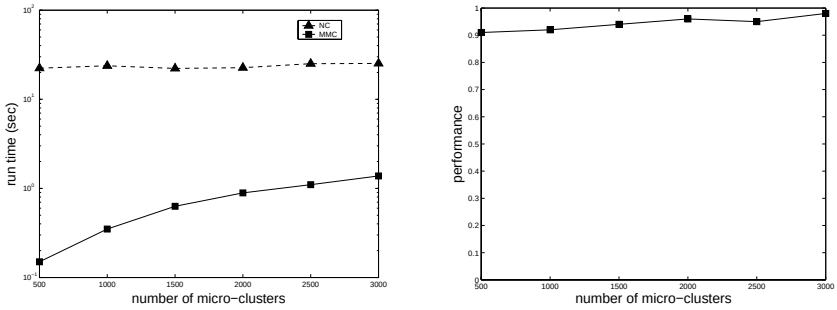
Given the vast volume of research work conducted on clustering and moving objects individually, it is a little surpris-

[3]Since the measure of NC is trivially 1, there is only single curve shown in each figure.



(a) Run time comparison (b) Performance of MMC (Perf(NC)≡1)

Figure 3: Different number of objects with datasets $K100N_{mmc}1000T30$.



(a) Run time comparison (b) Performance of MMC (Perf(NC)≡1)

Figure 4: Different number of micro-clusters with datasets of $K100N_l450N_u550T30$.

ing that little effort is seen to try to perform the clustering analysis on mobile data. To the best of our knowledge, the most related work to ours is [9], where a static clustering approach is presented. Basically, with the algorithm, the objects are clustered once in advance, and the clustering does not change with time. The author shows that by carefully choosing the representative objects, the resultant clustering is competitive at any time during the motion. However, although some theoretical bounds are given, the idea of fixed clustering does not appear attractive in real applications. Furthermore, a strong assumption is made there that no on-the-fly updates of objects are allowed, while they are supported by our algorithm.

7. CONCLUSION

In this paper, we explore the clustering analysis on moving objects, which is able to provide some interesting pattern changes and is of extensive interest. We propose the concept of moving micro-cluster to catch some regularities of moving object and handle the very large datasets. Efficient algorithms are proposed to keep the moving micro-clusters geographically small. Together with knowledge about the collisions among the moving micro-clusters, superb clustering results could be obtained, as shown in the experiments. With little modifications, our approach is expected to discover interesting clusters of various forms other than being geographically close.

8. REFERENCES

[1] M. Ankerst, M. Breunig, H. P. Kriegel, and J. Sander. OPTICS: ordering points to identify the clustering structure. *SIGMOD*, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD*, 1998.

[3] J. Bash, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *Journal of Algorithms*, 1999.

[4] J. Bash, L. J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. *SoCG*, 1997.

[5] D. Chudova, S. Gaffney, E. Mjolsness, and P. Smyth. Translation-invariant mixture models for curve clustering. *SIGKDD*, 2003.

[6] G. D. da Fonseca and C. M. H. de Figueiredo. Kinetic heap-ordered trees: tight analysis and improved algorithms. *Information Processing Letters*, 2003.

[7] A. Gersho and R. M. Gray. Vector quantization and signal compression. *Kluwer Academic*, 1992.

[8] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. *SIGMOD*, 1998.

[9] S. Har-Peled. Clustering motion. *FOCS*, 2001.

[10] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. *VLDB*, 2003.

[11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 1999.

[12] G. Karypis, E. H. Han, and V. Kumar. CHAMELEON: a hierarchical clustering algorithm using dynamic modeling. *COMPUTER*, 1999.

[13] Library for Efficient Data types and Algorithms. *<http://www.algorithmicsolutions.com/enleda.htm>*.

[14] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statist*, 1967.

[15] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. *VLDB*, 1994.

[16] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD*, 2000.

[17] Y. Tao and D. Papadias. Time-parameterized queries in spatial-temporal databases. *SIGMOD*, 2002.

[18] W. Wang, J. Yang, and R. Muntz. STING: a statistical information grid approach to spatial data mining. *VLDB*, 1997.

[19] O. Wolfson, P. A. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 1999.

[20] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. *SIGMOD*, 1996.