Online Clustering of Trajectory Data Stream

Ticiana L. Coelho da Silva
Federal University of Ceará, Brazil
Email: ticianalc@ufc.br

Karine Zeitouni
Université de Versailles-St-Quentin, France
Email: karine.zeitouni@uvsq.fr

José A. F. de Macêdo
Federal University of Ceará, Brazil
Email: jose.macedo@lia.ufc.br

***Abstract*—Movement tracking becomes ubiquitous in many applications, which raises great interests in trajectory data analysis and mining. Most existing approaches cluster the whole trajectories offline. This allows characterizing the past movements of the objects but not current patterns. Recent approaches for online clustering of moving objects location are restricted to instantaneous positions. Subsequently, they fail to capture moving objects' behavior over time. By continuously tracking moving objects' sub-trajectories at each time window, rather than just the last position, it becomes possible to gain insight on the current behavior, and potentially detect mobility patterns in real time. In this work, we tackle the problem of discovering and maintaining the density based clusters in trajectory data streams, despite the fact that most moving objects change their position over time. We propose CUTiS, an incremental algorithm to solve this problem, while tracking the evolution of the clusters as well as the membership of the moving objects to the clusters. Our experiments were conducted on real data sets, and it shows the efficiency and the effectiveness of our method.**

I. Introduction

The huge volume of collected trajectories opens new opportunities for discovering hidden mobility patterns. These patterns allow characterizing individual mobility as well as groups sharing similar trajectories for a certain time interval. Usually, this analysis is done off-line by applying data analysis and mining techniques on the previously collected data. This allows characterizing the past movements of the objects but not the current mobility patterns. Nowadays, many services involve tracking moving objects (e.g., persons, vehicles, animals) to report their trajectory continuously (e.g., every second or every minute). Analyzing these data while they are generated may bring a real added-value in the comprehension of the city dynamics, and the detection of regularities as well as anomaly, which is essential for decision making. Among these patterns, we consider in this paper the (sub)trajectory clustering and its evolution. Such discovery may help the search for effective re-engineering of traffic, or dynamically detecting events or incidents at a city level.

Tracking applications need to deal with the incremental nature of spatio-temporal data. Indeed, data arrive rapidly in a short period of time and its size keep growing as time goes. The exploration of the entire data stream might not be very useful since the information to be extracted may be outdated. Analyzing the data while it arrives can provide much better comprehension of current cluster patterns and their evolution between consecutive periods of time. By current cluster patterns, we mean the groups of moving objects having similar sub-trajectories during the last time window. Thus, by tracking sub-trajectories rather than just the last position, it becomes possible to gain insight on the current behavior, and potentially detect suspicious behaviors or remarkable events on the moment it happens. However, finding clusters over these data stream in (quasi) real time is quite challenging because all tracked moving objects may change their positions every time, and clusters may also change accordingly. Furthermore, new moving objects may appear as well as others may stop and disappear. These changes may affect clusters formation.

In this paper, we address the problem of online clustering discovery and its evolution by tracking moving objects at consecutive time windows. To solve this problem, we propose CUTiS (standing for ClUstering Trajectory Stream), an incremental algorithm tailored for trajectory data stream: (i) most the objects changes their sub-trajectories data at each time, (ii) new moving objects appear as well as (iii) others disappear from the system. Because we do clustering for each time window rather than for the whole trajectory, the clusters we find are actually sub-trajectory clusters. We define a new structure, called micro-group, to represent the relationship among moving objects. Then, micro-groups may evolve, e.g., merge or split in the next time period. In our experiments, the maintenance of micro-groups structure presents less computational cost than running the clustering algorithm from scratch.

There exist approaches for incremental clustering of moving objects position and detects its evolution [1], [2], [3], [4], [5], but they are restricted to instantaneous positions failing to capture mobility behavior along a time window. Besides, there are some approaches in the literature aiming at clustering trajectories as a whole [6], [7], [8], some clustering algorithms for sub-trajectories [9], [10] focusing on spatial criteria and ignoring the time dimension and some that are road network constrained [11], [12], [13]. According to the author's knowledge, we believe that our study is relevant, since there is no approach for discovering and updating clusters of trajectory data stream in (time, space and direction). The main contributions of CUTiS are as follows:

- A novel definition of micro-group that we believe enables the computation of interesting patterns.
- An incremental algorithm to maintain micro-groups and to capture its evolution patterns on trajectory data stream.
- A sub-trajectory clustering algorithm based on time, space and direction distance functions.

We conducted an extensive study on real data sets to evaluate the effectiveness and efficiency of CUTiS. This paper

is organized as follows: In the next section, we provide the problem statement. Section III presents our approach, and Section IV shows our experimental evaluation. Section V presents the main related works, before the conclusion.

II. Problem Definition

A trajectory is a sequence of the locations of a moving object at each time-stamp and is denoted by $TR_j = < p_1 p_2 ... p_r ... p_{len_j} >$. Here, p_k $(1 \leq k \leq len_j)$ is a point (x_k, y_k, t_k) in a three dimensional space, where (x_k, y_k) indicates the location of the object at time t_k. The length len_j of a trajectory can be different from those of other trajectories. In addition, CUTiS is time constrained, we use linear interpolation to align the trajectories in time. A trajectory $< p_{l_1}, p_{l_2}, ..., p_{l_k} >$ $(1 \leq l_1 < l_2 < ... < l_k \leq len_j$, where $l_k = l_{k-1}$+1) is called a sub-trajectory of TR_j.

Definition 1: **(Input Stream)** Let $i = [t, t + \delta t]$ be an observed time window. The set $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}),...,(o_n, ST_{n,i})\}$ is called *input stream* at time window i if each $ST_{j,i}$ is a sub-trajectory of a moving object o_j during the time window i, and there is no temporal overlap between different input streams.

We aim to track moving objects and discover sub-trajectory clusters incrementally using I_i avoiding executing clustering algorithm from scratch at each time window. Because our data are dynamic, each moving object sub-trajectory is manipulated in CUTiS as one of these operations: (i) A moving object can appear in the system (inserted into the system), (ii) a moving object can disappear from the system (deleted from the system), and also (iii) moving objects may update their sub-trajectories during the time window. These operations may affect the existing clusters.

Problem Statement: For each time window i, when the data stream $I_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_n, ST_{n,i})\}$ arrives in the system, the goal is to track the moving objects and update clusters avoiding re-computing them from scratch at each time window. We should discover sub-trajectory clusters with respect to movement similarity taking into account time, space and direction.

To discover sub-trajectory clusters with various shapes, we employ the concept of density-based clustering in this study [14].

III. Our Approach

CUTiS, firstly proposed in [15], follows four steps: apply a distance function, choose representative trajectories, maintain the micro-groups, and discover sub-trajectory clusters.

- **Apply a Distance Function**: To measure the distance between two moving objects' sub-trajectories $ST_{k,i}, ST_{j,i}$ at time window i, we implement the synchronous Euclidean distance, which accounts for time, space, and direction presented in Section III-A. However, CUTiS is suitable to any distance function for trajectories.
- **Choose Representative Trajectories**: For each group of moving object sub-trajectories $S_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_m, ST_{m,i})\}$ at the time window i, we define the representative trajectory as a pair composed by one moving object and its sub-trajectory which "better represents" the behavior of S_i. To choose $(o_j, ST_{j,i})$ as a representative sub-trajectory, CUTiS uses two metrics: the number of moving objects that have their sub-trajectory similar to $ST_{j,i}$ and a Gaussian kernel function to estimate the representativeness of $ST_{j,i}$. This is discussed in the Section III-B.
- **Maintain the Micro-groups**: We define a new structure in Section III-C, called micro-group, to capture and maintain small and dense groups of moving objects around the representatives at each time window. This adapts to many situations such as families staying together or persons sharing the same trip using a public transport. Furthermore, their maintenance cost is lower than maintaining a big cluster during each time window. The micro-group definition is based on the use of a representative trajectory. We propose an algorithm to incrementally maintain each micro-group and to capture its evolution patterns. We also propose density based cluster discovery by merging micro-groups.
- **Discover Sub-trajectory Clusters**: We use the maintained micro-groups to discover patterns, by capturing the evolution of micro-groups over time. Since each micro-group is density based, it is suitable to find sub-trajectory density based clusters (for example, merging micro-groups results in density based sub-trajectory clusters). We confirm in our experiments that CUTiS is more efficient than running a clustering algorithm from scratch at each time window (we used DBSCAN [14] as our baseline algorithm). This step is presented in Section III-D.

A. Distance Function

The distance function used in CUTiS has two levels: time and space, and direction level.

Definition 2: **(Spatio-Temporal Distance)** For the time window $i = [t, t + \delta t]$, the distance between two moving objects' sub-trajectories $ST_{j,i}$ and $ST_{k,i}$ is computed by the synchronous distance $D_{ST_{j,i}, ST_{k,i}}$, as follows [6]:

$$dist_\lambda(ST_{j,i}, ST_{k,i}) = \frac{\int_t^{t+\delta t} D_{ST_{j,i}, ST_{k,i}}(t)\,\mathrm{d}t}{\delta t} \quad (1)$$

Let $D_{ST_{j,i}, ST_{k,i}}(t)$ be the distance function between sub-trajectories $ST_{j,i}, ST_{k,i}$ at time t. Considering the time interval $[t, t + \delta t]$ and n be the number of timestamps of position updates. In case the sampling rates of sub-trajectories vary, the locations of missing updates are approximated by linear interpolation between consecutive recorded points, so (1) is adapted to:

$$dist_\lambda(ST_{j,i}, ST_{k,i}) = \frac{\sum_{r=1}^{n-1} \int_{t_r}^{t_{r+1}} D_{ST_{j,i}, ST_{k,i}}(t)\,\mathrm{d}t}{\delta t} \quad (2)$$

where each t_r is the timestamp when at least one of the moving objects report its position. The Euclidean distance between two objects that move with linear functions of time between consecutive timestamps, was defined in [16] as:

$$D_{ST_{j,i},ST_{k,i}}(t) = \sqrt{at^2 + bt + c} \quad (3)$$

where a, b, c are the factors of this trinomial ($a \geq 0$). As proved in [17], the integral can be computed on $O(1)$ and can be efficiently computed with Trapezoid Rule, [17] provided also bounds for the approximation error:

$$dist_\lambda(ST_{j,i}, ST_{k,i}) \approx \frac{\frac{1}{2}\sum_{r=1}^{n-1}[D_{ST_{j,i},ST_{k,i}}(t_r)] * (t_{r+1} - t_r)}{\delta t} + \frac{\frac{1}{2}\sum_{r=1}^{n-1}[D_{ST_{j,i},ST_{k,i}}(t_{r+1})] * (t_{r+1} - t_r)}{\delta t} \quad (4)$$

Definition 3: **(Direction Distance)** The distance between two moving objects sub-trajectory $ST_{j,i}, ST_{k,i}$ at direction level, during a time window $i = [t, t+\delta t]$ is based on the angle between the sub-trajectories line segments. As we consider the time discretized, the direction distance is computed by the sum of a function D_θ between two line segments of $ST_{j,i}$ and $ST_{k,i}$ on the same time interval:

$$dist_\theta(ST_{j,i}, ST_{k,i}) \approx \frac{\sum_{r=1}^{n-1} D_\theta[(ST_{j,i}(t_r), ST_{j,i}(t_{r+1})), (ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))]}{\delta t} \quad (5)$$

where $(ST_{j,i}(t_r), ST_{j,i}(t_{r+1}))$ and $(ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))$ are line segments of the sub-trajectories $ST_{j,i}$ and $ST_{k,i}$, respectively, and t_r is, as above, a timestamp when an update holds. We defined D_θ based on [10] and it expresses the maximum distance that a moving object of line segment minimum length will cover during rotation. Let $L_j = (ST_{j,i}(t_r), ST_{j,i}(t_{r+1}))$ and $L_k = (ST_{k,i}(t_r), ST_{k,i}(t_{r+1}))$, and $e = min(|L_j|, |L_k|)$.

$$D_\theta(L_j, L_k) = \begin{cases} e * \sin(\theta), & \text{if } 0\,^\circ \leq \theta < 90\,^\circ. \\ e, & 90\,^\circ \leq \theta \leq 180\,^\circ. \end{cases} \quad (6)$$

The angle θ is calculated using vector operations. Let $\vec{ab}$ denote a vector constructed by two points a and b. As L_j and L_k are line segments and composed by two points, we discover θ as follows:

$$\cos(\theta) = \frac{\vec{L_j}\vec{L_k}}{\|\vec{L_j}\|\|\vec{L_k}\|} \quad (7)$$

We finally define the distance measure between two moving objects sub-trajectories $ST_{j,i}$ and $ST_{k,i}$. The weights ω_θ and ω_λ are determined depending on the application. We set these weights equally to 0,5 as default.

$$distance(ST_{j,i}, ST_{k,i}) = \omega_\theta * dist_\theta(ST_{j,i}, ST_{k,i}) + \omega_\lambda * dist_\lambda(ST_{j,i}, ST_{k,i}) \quad (8)$$

B. Choosing a Representative Trajectory

From a group of moving object sub-trajectories, the representative trajectory tries to describe the overall movement.

Definition 4: **(Representative Trajectory)** Consider a set of moving object sub-trajectories $S_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), .., (o_m, ST_{m,i})\}$ at the time window i. Let ρ be a representativeness threshold, σ be a standard deviation (sometimes called the Gaussian width), ϵ be a given distance threshold and τ be a size/density minimum threshold. Then, $(o_j, ST_{j,i})$ is a *representative trajectory* of S_i if and only if:

- $\forall (o_k, ST_{k,i}) \in S_i$, $\text{voting}(ST_{j,i}, ST_{k,i}) = e^{-\frac{distance^2(ST_{j,i}, ST_{k,i})}{2\sigma^2}} > \rho$
- $N_\epsilon(o_j) = \{(o_k, ST_{k,i}) \in S_i | distance(ST_{j,i}, ST_{k,i}) \leq \epsilon\}, then\, |N_\epsilon(o_j)| \geq \tau$

Our "voting" function is based on [18] which uses the Gaussian function to quantify the representativeness of trajectory line segments. In this work, it quantifies the representativeness of a sub-trajectory for a group of moving object sub-trajectories. In order to be representative, the "voting" of a sub-trajectory should be significant (greater than ρ threshold). One idea to choose ρ is to establish the maximum distance "allowed" from one moving object sub-trajectory to its representative. Moreover, in CUTiS we also choose the representative trajectory based on its density property (dense neighborhood).

The intuition behind the relationship of "distance" and "voting" function is: If "distance" is close to zero, the "voting" is close to its maximum value. This means that if $ST_{j,i}$ is very close (in time, space and direction, for example) to $ST_{k,i}$, then $(o_j, ST_{j,i})$ is a candidate to be the representative of $ST_{j,i}$. Otherwise, if the "distance" is high, the "voting" function is close to its minimum value, meaning that $ST_{j,i}$ is very far way from $ST_{k,i}$, so $ST_{k,i}$ can be not represented by $ST_{j,i}$.

We do not use centroid (as in the related works [19], [20]) instead of representative trajectory, because we handle sub-trajectory data. It is not trivial to estimate the centroid sub-trajectory (according to space, time and direction similarity), moreover we may have different sampling rates for each sub-trajectory during a time window.

C. Micro-group Definition and Maintenance

The following section presents a new structure called micro-group, based on the concept of representative trajectory.

Definition 5: **(Micro-Group)** Let $S_i = \{(o_1, ST_{1,i}), (o_2, ST_{2,i}), ..., (o_m, ST_{m,i})\}$ be a set of moving object sub-trajectory at time window i, let O_i be the set of moving objects in S_i, ϵ be a distance threshold, τ be a size/density minimum threshold, ρ be a representativeness threshold, a *micro-group* g is defined as a set of objects satisfying:

1) $g \subseteq O_i$
2) $\exists o_j \in g$, such that $R_g^{traj} = (o_j, ST_{j,i})$ is a representative trajectory of g w.r.t. ϵ, τ and ρ.

Consider $R_g^{traj} = (o_j, ST_{j,i})$, we use $object[R_g^{traj}]$=o_j and $traj[R_g^{traj}]$=$ST_{j,i}$. To create a micro-group, the representative trajectory is randomly chosen among the core objects (as defined in DBSCAN). Then, the micro-group is derived as the objects that vote for the chosen representative as we show on the Algorithm 1. The initialization cost is $O(n^2)$ time for n moving objects. However this step only needs to be carried out once and each micro-group is dynamically maintained along the stream for each time window according to the Algorithm 2. Furthermore, the cost $O(n^2)$ cannot be reduced, since moving objects are always changing their positions in time and maintain spatial index (such as 3D R-tree) at each time window may incur high cost.

The Algorithm 1 randomly picks an unvisited moving object sub-trajectory $(o_j, ST_{j,i})$ and checks if it can be a representative trajectory. If it has a density neighborhood (according to ϵ radius and τ threshold (Lines 6-7)), a new micro-group "g" is created with o_j and its neighbors that can be represented by $(o_j, ST_{j,i})$ (Lines 7-12). Otherwise, o_j is marked as ungrouped (Lines 23-24), because it can to be represented by one another representative and then belong to an existing micro-group. Or it can be an outlier (Lines 26-27). On Lines 15-19, the Algorithm tries to expand the micro-group g, inserting moving objects which are not visited yet, and can be represented by $(o_j, ST_{j,i})$. On Line 21, $(o_j, ST_{j,i})$ is finally defined as representative trajectory of micro-group g. The Algorithm 1 outputs a set of micro-groups (Line 28) from a moving object sub-trajectory set.

Since CUTiS deals with data stream, the most important contribution is the maintenance phase. The intuition behind is: (i) to check for each micro-group whether the representative is still valid in the next time window (it survives), otherwise, the micro-group disappears or splits, (ii) to track moving object sub-trajectories that are likely to join the micro-group (e.g., outliers, new objects, and other micro-group objects migrating to another micro-group), (iii) for the remaining objects, a similar process to the initialization allows creating new micro-groups. Since the Algorithm 1 is not deterministic, the interest of maintenance is two folds: (i) by running from scratch the initialization algorithm leads to totally different micro-groups, (ii) and does not allow to track the evolution of micro-groups. CUTiS captures the evolution patterns in the maintenance phase, which provides insights about the nature of cluster/group changes [21].

The Algorithm 2 maintains a micro-group given as input, let g_{i-1} be it. The Algorithm 2 receives the set of moving objects that should be deleted (g_i^{old}) from g_{i-1} and the set of moving objects that update their sub-trajectories (g_i^{upd}). We consider that $g_{i-1} \subseteq g_i^{old} \cup g_i^{upd}$ and $g_i^{old} \cup g_i^{upd} \subseteq g_{i-1}$, i.e. the moving object that is not removed from g_{i-1}, so it updates its sub-trajectory. For the new moving objects, the Algorithm 3 tries to find an existing micro-group to add them. In this way,

Algorithm 1: Initialize micro-group

Input: S_i moving object sub-trajectory set for the time interval $i = [t, t + \delta t]$, ϵ distance thresholds, τ the size threshold, ρ representativeness threshold
Output: G_i: a set of micro-group

```
1  begin
2      G_i ← empty;
3      mark all the sub-trajectories in S_i as unvisited;
4      while ∃(o_j, ST_{j,i}) ∈ S_i unvisited do
5          mark (o_j, ST_{j,i}) as visited;
6          get N_ε(o_j);
7          if |N_ε(o_j)| -1 ≥ τ then
8              for o_k ∈ N_ε(o_j) not visited do
9                  vote ← voting(ST_{j,i}, ST_{k,i});
10                 if vote > ρ then
11                     g ← g ∪ {o_k};
12                     mark (o_k, ST_{k,i}) as visited;
13
14
15             for (o_k, ST_{k,i}) not visited do
16                 vote ← voting(ST_{j,i}, ST_{k,i});
17                 if vote > ρ then
18                     g ← g ∪ {o_k};
19                     mark (o_k, ST_{k,i}) as visited;
20
21             R_g^{traj} ← (o_j, ST_{j,i});
22             G_i ← G_i ∪ {g};
23         else
24             mark ST_{j,i} as ungrouped;
25
26     for ST_{j,i} ungrouped do
27         mark o_j as outlier;
28     return G_i;
```

we can see that CUTiS is suitable for data stream, because it handles insertions, deletions and consider that the remaining moving objects update their sub-trajectories, unlike [4], [19], [20], [22], among others.

The Lines 2-3 in the Algorithm 2, g_{i-1} is updated and relabels to g_i. At this point, as the micro-group data changes, it is necessary to check if $(o_j, ST_{j,i}) = R_{g_i}^{traj}$ continuous to be the representative trajectory of g_i. In the Line 6, the variable $size$ stores the number of o_j's neighbors (ϵ radius) and the algorithm checks the neighbors vote (Line 7). If $(o_j, ST_{j,i})$ continuous to be the representative of g_i, w.r.t. τ, ϵ and ρ, then g_i survives (Lines 8-9). The algorithm checks for each moving object $o_k \in g_i$, if it can still be represented by $(o_j, ST_{j,i})$ (Lines 10-13) or it might migrated to another micro-group (Lines 14-19). In this last case, o_k is deleted from g_i and migrated to another micro-group(Lines 14-17), otherwise it becomes an outlier (Lines 18-19). If g_i changes

Algorithm 2: Incremental Maintenance of a micro-group

Input: g_{i-1}: micro-group, ϵ distance thresholds, τ the size threshold, ρ representativeness threshold, $\{g_i^{old}, g_i^{upd}\}$ are sets of moving objects to be deleted,updated from g_{i-1} to g_i

Output: updated micro-group g_i

1 **begin**
2 $\forall o_j \in g_i^{upd}$: updates its sub-trajectory in g_{i-1}
3 $g_i \leftarrow g_{i-1} - g_i^{old}$
4 Let $(o_j, ST_{j,i}) = R_{g_i}^{traj}$;
5 get $N_\epsilon(o_j)$;
6 $size \leftarrow |N_\epsilon(o_j)|$;
7 **if** $size \geq \tau$ *and* $\forall o_k \in N_\epsilon(o_j)$ $voting(ST_{j,i}, ST_{k,i}) > \rho$ **then**
8 g_i **survives**;
9 $(o_j, ST_{j,i})$ continues to be the representative trajectory for g_i;
10 **foreach** $o_k \in \{g_i \setminus N_\epsilon(o_j)\}$ **do**
11 $vote \leftarrow voting(ST_{j,i}, ST_{k,i})$;
12 **if** $vote > \rho$ **then**
13 o_k continuous to belong to g_i
14 **else**
15 Delete o_k from g_i;
16 **if** $\exists g_z \in G_i$, $voting(traj[R_{g_z}^{traj}], ST_{k,i}) > \rho$ **then**
17 Insert o_k into g_z;
18 **else**
19 o_k is an outlier;
20
21 **else**
22 **if** $|g_i| \geq \tau$ **then**
23 g_i **splits**;
24 $S_{g_i} \leftarrow$ set of g_i moving objects sub-trajectory;
25 $initializeMicroGroup(S_{g_i}, \epsilon, \tau, \rho)$;
26 **else**
27 g_i **disappears**
28 $\forall o_k \in g_i, o_k$ is an outlier;
29 **return** g_i

its representative trajectory, it means that the micro-group's behavior changes, then g_i splits or disappears from the system. In the case that it splits, it is necessary to rebuild g_i into one or more micro-group(s) using g_i data (Lines 21-25). However, if g_i is not density enough to generate micro-group(s), its moving objects become outliers (Lines 27-28) and g_i disappears. We only merge two or more micro-group when they can form a cluster, since by merging micro-groups might not result in another micro-group (imagine when there is no trajectory to represent all merging micro-groups objects). Let m be the number of micro-groups at time window i and n_u be the number of moving objects which update their sub-trajectories. The maintenance cost w.r.t. Algorithm 2 is $O(m * n_u)$, "headed" by the cost in the Lines 10-20. As Algorithm 2 is called for each micro-group, finally the total cost is $O(m^2 * n_u)$. As the number of micro-groups is much smaller than the number of moving objects, the incremental maintenance is still more attractive than initialize micro-groups from scratch.

The Algorithm 3 tries to find for each new moving object, a micro-group g_i (Line 3), such that its representative trajectory can represent the new moving object sub-trajectory. Let $(o_k, ST_{k,i})$ be a new one, if $voting(traj[R_{g_i}^{traj}], ST_{k,i}) > \rho$ (Line 4), then o_k can be inserted into g_i (Line 6). The remaining objects that cannot be assigned to any existing micro-group might create new ones (Line 9), including the Outliers set. The cost analysis did for Algorithm 1 can be applied to Algorithm 3, by taking into account the number of new moving objects and outliers at time window i.

A micro-group can disappear or split because of moving object(s) deletion(s) or update(s), however by inserting new moving object(s) the micro-group may survive. We leave thus issue for future work.

Algorithm 3: Insert new moving objects

Input: G_i: the set of micro-groups, ρ the representativeness threshold, I_i^{new}: a set of new moving objects sub-trajectories from the time window i

1 **begin**
2 **foreach** $\{o_k, ST_{k,i}\} \in I_i^{new}$ **do**
3 **foreach** $g_i \in G_i$ **do**
4 $vote \leftarrow voting(traj[R_{g_i}^{traj}], ST_{k,i})$;
5 **if** $vote > \rho$ **then**
6 $g_i \leftarrow g_i \cup \{o_k\}$;
7 Remove $(o_k, ST_{k,i})$ from I_i^{new};
8 **break**;
9 $initializeMicroGroup(I_i^{new} \cup Outliers, \epsilon, \tau, \rho)$;

D. Density-based Sub-trajectory Clustering

Some existing clustering algorithms have to check the density connectivity for each object which makes the approach computational costly. CUTiS avoids computing all the moving objects density connections, and uses the representative trajectory in the clustering process, since it tries to better describe the behavior of a micro-group. Fortunately, each micro-group is density based, it is suitable to find sub-trajectory density based clusters by merging two or more micro-groups into one cluster. In this way, we propose a clustering algorithm which is a variant of DBSCAN based on two parameters ϵ and τ. It has two phases which are described below and presented on the Algorithm 4.

Choose the merge candidates to form a cluster. This first phase aims to find the micro-groups candidates to be

merged into a cluster. Imagine that each micro-group g_i has a coverage area on the format of circular shape. This area is derived from g_i radius which is the distance value between g_i representative trajectory and the farthest moving object's sub-trajectory in g_i. Two micro-groups are candidates to merge into a density based cluster, if they intersect or are tangent to each other w.r.t. their circular area. Let g_i and g_k be micro-groups and r_{g_i} and r_{g_k} their respectively radius. If $distance(traj[R^{traj}_{g_i}], traj[R^{traj}_{g_k}]) \leq r_{g_i} + r_{g_k}$ is satisfied, then g_i and g_k are merge candidates.

Algorithm 4: Sub-trajectory Clustering

Input: G_i: the set of micro-groups, ϵ distance threshold, τ the size threshold
Output: C: is a set of clusters

```
1  begin
2      while G_i ≠ ∅ do
3          randomly pick a micro-group g_i from G_i;
4          Initialize cluster c ← g_i, add c to C;
5          remove g_i from G_i;
6          foreach g_i unvisited in c do
7              mark g_i as visited;
8              foreach g_k ∈ MergeCandidates(g_i, ε, G_i)
               do
9                  foreach o_i ∈ g_i, o_k ∈ g_k do
10                     if distance(o_i, o_k) ≤ ε then
11                         n_i ← |N_eps(o_i)|;
12                         n_k ← |N_eps(o_k)|;
13                         if n_i ≥ τ and n_k ≥ τ then
14                             c ← c ∪ g_k;
15                             remove g_k from G_i;
16                             break;
17     return C;
```

Density Based Clustering. In the second phase, CUTiS finally finds the clusters using the merge candidates. By pruning the merge candidates, CUTiS saves computation and the final clustering result is approximated to the one which is produced by applying the original DBSCAN w.r.t. ϵ and τ as presented in the experiments section. The following definitions address the concept of density connectivity in this work.

Definition 6: **(Direct Density Reachable)** Let G_i be the set of micro-groups at time window i. A micro-group $g_i \in G_i$ is directly density reachable from a micro-group $g_k \in G_i$ w.r.t. ϵ and τ, if $\exists o_i \in g_i, o_k \in g_k$, such that the $distance(o_i, o_k) \leq \epsilon$, $|N_\epsilon(o_i)| \geq \tau$ $(N_\epsilon(o_i) = \{o'_i \in G_i | distance(o_i, o'_i) \leq \epsilon\})$ and $|N_\epsilon(o_k)| \geq \tau$ $(N_\epsilon(o_k) = \{o'_k \in G_i | distance(o_k, o'_k) \leq \epsilon\})$.

Definition 7: **(Density Reachable)** A micro-group g_i is density reachable from g_k w.r.t. ϵ and τ, if there is a chain of micro-groups $\{g_{s_1}, ..., g_{s_n}\}$, where $g_{s_1} = g_i$, $g_{s_n} = g_k$ such that $g_{s_{j+1}}$ is directly density reachable from g_{s_j}.

Definition 8: **(Density Connected)** A micro-group g_i is density connected to g_k w.r.t. ϵ and τ, if there is a micro-group g_j such that g_i and g_k are density reachable from g_j w.r.t. ϵ and τ.

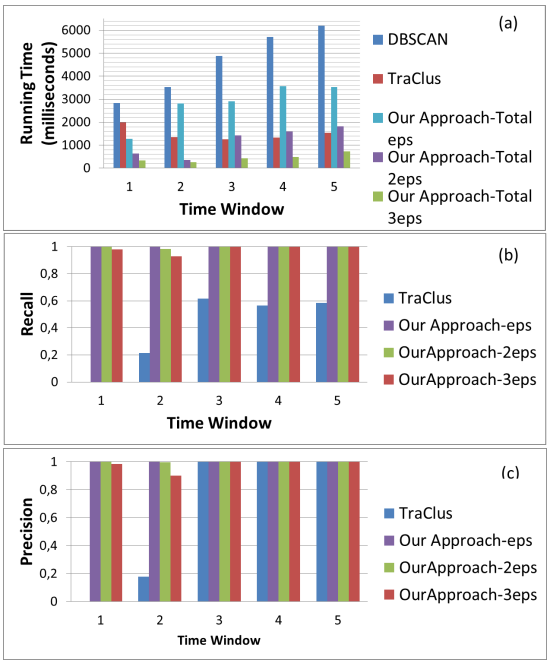

Figure 1: Analysis of Taxi data set from Fortaleza: (a) Total Running Time for 5 time windows, (b) Recall and (c) Precision for clusters detection.

The Algorithm 4 firstly picks a micro-group and initialized it as a new cluster (Lines 4-5). For each micro-group g_i in a cluster, the algorithm checks its candidates to merge (Line 8), some micro-groups are filtered out. For each micro-group g_k candidate to merge with g_i, the algorithm checks if they are direct density reachable (Lines 9-13). If they are, the algorithm merges them into the same cluster(Lines 14-15). The algorithm outputs the density connected clusters when all micro-groups belonged to any cluster (Line 17).

IV. Experiments

To the best of our knowledge, there is no other incremental trajectory clustering algorithm for trajectory data stream. However, for validation purpose, we employ as a baseline some methods of the state of the art: TraClus [10] and DBSCAN, with the following changes: (i) at each time window we run DBSCAN and TraClus given as input only the sub-trajectories of the current time window, and not the complete trajectory; (ii) we also adapted TraClus and DBSCAN to use our distance function, in order to take into account the distance in time, space and direction. We avoid to compare with [1] because to apply incremental clustering for each moving object trajectory at each time window is expensive [5].

Evaluation Metrics. The DBSCAN's results are used as the ground truth to test the effectiveness (recall and precision) because (i) DBSCAN has been regarded as the most representative density-based clustering algorithm and (ii) both TraClus

and CUTiS share many characteristics with the algorithm DBSCAN. The effectiveness is measured according to the found clusters and outliers detection. In our case, the recall measure for outliers detection is the proportion of correctly classified as outliers (true positives) over DBSCAN's outliers (true positives+false negatives). And the precision measure is the proportion of correctly classified as outliers (true positives) by CUTiS over all the objects classified as outliers (true positives+false positives) by CUTiS. Let Out_i^{dbscan} and Out_i^{mg} be the a set of moving object outliers found by DBSCAN and by CUTiS on the time window i, respectively. The recall and precision for outliers detection are measured as follows:

1) $Recall_{outlier} = \frac{|Out_i^{dbscan} \cap Out_i^{mg}|}{|Out_i^{dbscan}|}$
2) $Precision_{outlier} = \frac{|Out_i^{dbscan} \cap Out_i^{mg}|}{|Out_i^{mg}|}$

The recall and precision were also measured for the found clusters. At first, we matched each cluster C_i^{mg} found by CUTiS with only one DBSCAN cluster C_i^{dbscan}. On the matching process, we applied Jaccard Similarity to compare the clusters based on their moving objects labeled as core (since the set of core moving objects is a deterministic result in DBSCAN). In this way, the cluster C_i^{mg} matches with one DBSCAN cluster (for example, C_i^{dbscan}) that presents maximum Jaccard similarity value. The recall and precision are applied for each matched pair to measure the quality of clusters result. The recall for $\{C_i^{mg}, C_i^{dbscan}\}$ is the proportion of correctly core moving objects classified by CUTiS in C_i^{mg} (true positives) over the core moving objects classified by DBSCAN in C_i^{dbscan} (true positives+false negatives). The precision measure for $\{C_i^{mg}, C_i^{dbscan}\}$ is the proportion of correctly classified as core moving objects by CUTiS in C_i^{mg} (true positives) over all the core moving objects classified in C_i^{mg} by CUTiS (true positives+false positives). If two or more clusters of CUTiS match with the same DBSCAN cluster, we gather them in the same matching. In this paper, we reported the average recall and precision for all the matched clusters. Let $Core_i^{dbscan}$ and $Core_i^{mg}$ be the set of core moving objects in C_i^{dbscan} and C_i^{mg} clusters, respectively.

1) $Recall_{\{C_i^{mg}, C_i^{dbscan}\}} = \frac{|Core_i^{dbscan} \cap Core_i^{mg}|}{|Core_i^{dbscan}|}$
2) $Precision_{\{C_i^{mg}, C_i^{dbscan}\}} = \frac{|Core_i^{dbscan} \cap Core_i^{mg}|}{|Core_i^{mg}|}$

Meanwhile, we also measured the efficiency of CUTiS against our competitors. We implemented CUTiS and the baselines in Java, and tested them by using real data sets. The clustering parameters $eps = \epsilon$ and $minPoints = \tau$ are the same for CUTiS, DBSCAN and TraClus implementation. We set these parameters according to different data sets. In all experiments, we studied the influence of micro-group radius (and also the representativeness threshold ρ, they are related). The micro-group radius is varied according to the values eps, $2eps$ and $3eps$.

A. Experiments Results for Taxi Data Set from Fortaleza

This data set is computed from GPS recorded data of a taxi fleet in Fortaleza city, Brazil. It has around 400 trajectories of taxis. The taxi fleet belongs to a private transportation company that tags and tracks its vehicles as they move around, and aims to support drivers' decisions related to displacement through the city in (quasi) real time (i.e., movement behaviors through the city that can be found by trajectory clusters analysis). In the Figure 1, we evaluated the efficiency and effectiveness to cluster sub-trajectories using CUTiS, DBSCAN and TraClus algorithm. We set the time window size as 5 (five) minutes and the clusters are tracked for 5 (five) sequential time windows.

Note that CUTiS outperforms DBSCAN algorithm for all tested values of micro-group radius. When (i) micro-group radius=eps, CUTiS is from 20% a 54% faster than DBSCAN, for (ii) radius=$2eps$, it is from 70% to 90% faster and for (iii) radius=$3eps$, it is from 88% to 92% faster than DBSCAN. When micro-group radius increases, CUTiS also outperforms TraClus, because in this case there are less micro-group to be maintained. Even if TraClus outperformed CUTiS when radius=eps, the obtained clusters do not match with those of DBSCAN (our ground truth). The quality of our clusters is higher than TraClus as presented in the Figures 1(b) and 1(c). We omitted the effectiveness analysis for outliers detection due to the lack of space. In the next section, we will discuss in detail the experiments on a much larger number of moving objects, for better illustration of the advantages of CUTiS.

B. Experiments Results for Taxi Data Set from Beijing

This data set contains a real world GPS recorded data from taxis of Beijing [23]. We have used a trajectory subset of this data set containing around 7,000 trajectories referring to taxi travels around Beijing city. In the experiments, we set the time window size as 10 (ten) minutes and the clusters are tracked for 5 (five) sequential time windows.

Effectiveness Analysis of the Clustering approach. The effectiveness analysis is measured by applying the precision and recall metrics introduced above, and compares the shapes of the clusters in DBSCAN, TraClus and CUTiS where the clustering is computed by merging micro-groups. We expected that our method finds more clusters than DBSCAN, given the fact that the merge of two or more micro-groups only compares their representatives trajectories (this allows us to save the cost of pairwise sub-trajectory distance computation in DBSCAN). So each DBSCAN cluster is expected to match with many clusters from CUTiS. Actually, in our experiments, CUTiS found the same clusters as DBSCAN when the micro-group radius=eps and micro-group radius=$2eps$ (except very few differences).The Figure 2 compares the shapes of the clusters in DBSCAN, TraClus and CUTiS. The Figure concentrates in only two time windows and each color (blue, red, purple, green and black) represents one DBSCAN cluster. We obtain the best matches with DBSCAN, in both time windows, especially with radius=eps.

The differences are greater when micro-groups have radius=$3eps$. In the third time window, CUTiS included the clusters represented by green and red colors into the same cluster (represented by blue color). This means that a cluster

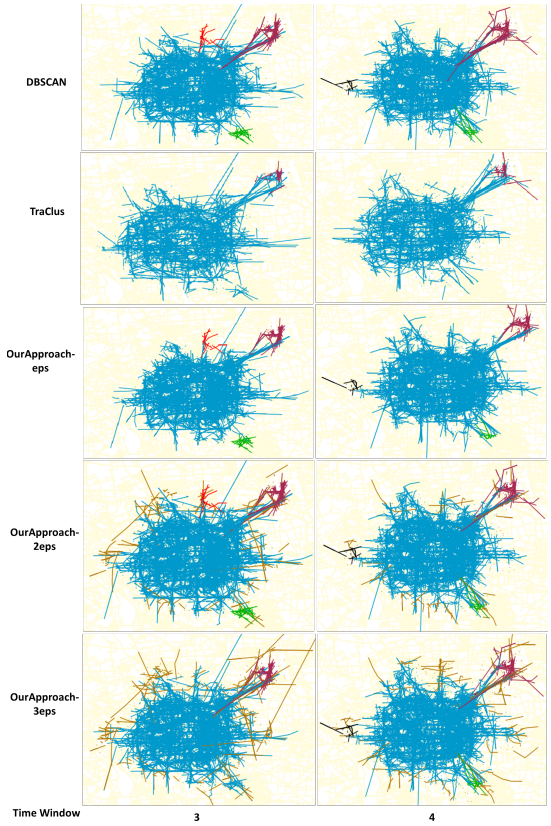

Figure 2: Comparison between clusters shapes of DBSCAN, TraClus and CUTiS (varying the micro-group radius) for 2 time windows using Taxi data set from Beijing

found by CUTiS contains core objects from three different DBSCAN clusters. This happens because these core objects can be represented by their micro-group representative trajectory, however the core objects are not density reachable objects (according to DBSCAN definition) to be in the same cluster. TraClus did not produce similar clusters to DBSCAN for both time windows, the clusters represented by red, green and black colors were not found. Furthermore, there are many moving objects in DBSCAN clusters that were not clustered by TraClus. The main reason for the difference between TraClus and DBSCAN results is the preprocessing phase in TraClus which partitions the trajectories which modifies the similarity since it is applied per partition instead of the whole sub-trajectories.

As expected, CUTiS presents higher precision and recall for cluster detection when the micro-group radius=eps and micro-group radius=2eps (Figure 3), since the clusters produced by CUTiS, in these cases, have similar shapes to DBSCAN clusters. Unlike when the micro-group radius=3eps and for TraClus approach (as we previously discussed). In the Figure 2, we highlighted with brown color the misclassified outliers by CUTiS. When micro-group radius becomes larger, it means that the representativeness threshold is higher, then some outlier sub-trajectories have enough votes to be

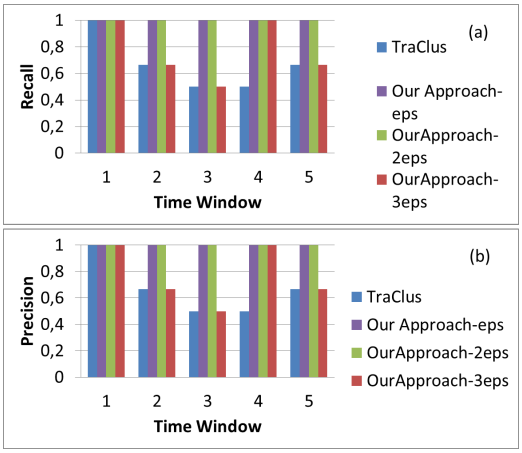

Figure 3: Effectiveness Analysis: (a) Recall and (b) Precision for detecting clusters using Taxi data set from Beijing

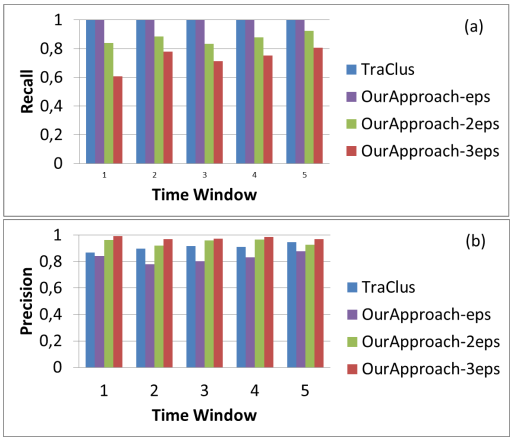

Figure 4: Effectiveness Analysis: (a) Recall and (b) Precision for detecting outliers using Taxi data set from Beijing

reached by one micro-group representative trajectory. This holds for radius=2eps and micro-group radius=3eps. In this case, CUTiS may misclassify outliers, and the recall for outliers detection decreases (Figure 4(a)). As expected, when the micro-group radius=eps, CUTiS does not misclassify outliers (since each micro-group only contains the representative trajectory which is a core object according to DBSCAN definition, and its directly density reachable sub-trajectories). However, there are some false positive outliers as well as in TraClus result. This affects the precision measure showed in the Figure 4(b). So when micro-group radius increases, CUTiS may misclassify outliers but also brings down the number of false positive outliers. This means the recall decreases and the precision increases for outliers detection. This trade-off between precision and recall in outlier detection depends on the application needs. However we believe that CUTiS is quite satisfactory for both measures since it does not present very low values.

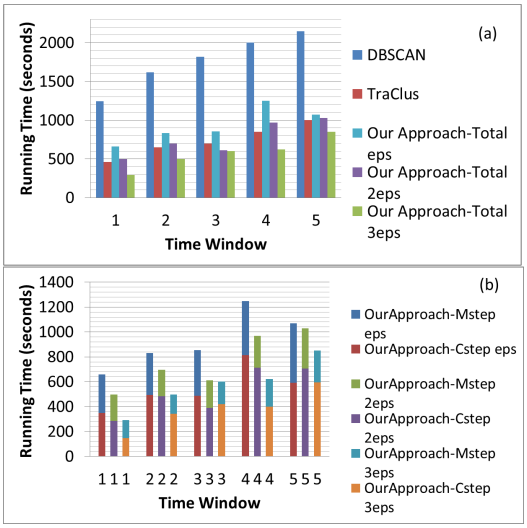

Figure 5: Efficiency Analysis: (a) Total Running Time for 5 time windows using Taxi data set from Beijing and (b) Running Time of Micro-group maintenance and clustering

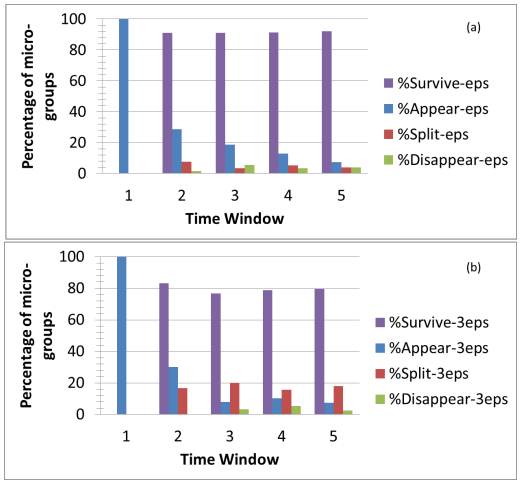

Figure 6: Micro-group evolution pattern for (a) radius=eps and (b) radius=3eps on the 5 tracked time windows using Taxi data from Beijing

Efficiency Analysis of the Clustering approaches. Data stream applications impose a limited memory constrained, it becomes difficult to provide arbitrary-shaped clustering results using conventional algorithms as DBSCAN. It is unrealistic to provide such a precise result. In the original DBSCAN, the system has to check the density connectivity for each moving object sub-trajectory which makes the approach computational costly (showed in Figure 5(a)). To discover density based clusters from trajectory data stream, CUTiS has two steps namely the creation and maintenance step (Mstep, Algorithms 1, 2 and 3), and clustering step (Cstep, Algorithm 4). In the Figure 5(b), the clustering step dominates the total cost in most of the time windows. Since to merge micro-groups into a cluster, we need to detect if they are density connected which is expensive (in the worst case, it is necessary to compare all pairwise moving objects that belong to these micro-groups). However, the micro-group structure can save computation in the maintenance processing avoid accessing all data set objects in details. Our results present more gain in efficiency than DBSCAN when micro-group radius increases (it maintains less representative trajectories, i.e. less number of micro-groups). As we mentioned before, TraClus approximated in the experiments most of the time the sub-trajectory to only one segment. In this case, the distance computation is less costly than consider each sub-trajectory with many segments as CUTiS did. This is the main reason why TraClus is much faster to do clustering. However, it only essentially outperformed CUTiS when micro-group radius=eps.

In general, our results present a trade-off between quality and performance, it is influenced by the micro-group radius value. Generally speaking, when the radius has low value, i.e. the micro-groups size decreases, it leads to maintain too many representatives (low performance) but our clustering algorithm produces similar result to DBSCAN (high quality). By increasing the micro-group radius, CUTiS presents high performance but it may lead to misclassify outliers (low quality).

Micro-group Evolution Pattern. CUTiS captures the micro-group evolution. The Figure 6 reports the percentage of micro-groups that evolve according to each pattern for all the tracking time windows. One can clearly see that the proportion of unchanged micro-groups (micro-groups that survive) is greater when micro-group radius is lower. The intuition is when the radius is lower, there are more micro-groups, and consequently it is necessary to choose more representatives. The probability to randomly choose good ones is greater than when micro-group radius increases (less representatives trajectories are chosen). A micro-group splits when its representative trajectory does not have enough votes to remain the representative in the current time window. There is less micro-groups that split when radius=eps (CUTiS got more "good" ones representatives) than when micro-group radius=3eps.

Hence we suggest that in real applications, the user should set higher micro-group radius to achieve higher performance to cluster sub-trajectories. However, if the priority is to have quality in the results (considering DBSCAN as the ground truth), we suggest to set lower value for micro-group radius in order to filter out outliers, and to guarantee the algorithm's sensitivity.

V. Related Work

In this section, we review existing works in the domains related to ours. In our setting, tracking the object movements and incrementally maintaining the sub-trajectory density clusters from trajectory data stream is a relevant problem that was not addressed before.

Trajectory Clustering There are some works related to clustering trajectory data [6], [7], [8] however the major problem is those approaches tend to generate clusters for

the entire trajectory data set, instead of the most recent time window. Hence the fine-grained spatio-temporal relationships between moving objects are lost. The papers [10], [11], [12] cluster sub-trajectory data, but [10] focused on spatial criteria and ignored the time dimension. And [11], [12] only consider road network constrained movement. Also they are not suitable for incremental data since clusters are re-calculated from scratch every time. The work [22] proposes efficient algorithms for maintaining and updating the clusters when new trajectories are received. However it does not consider the temporal aspects of the trajectories. As such, moving objects whose trajectories are in the same cluster may not actually stay together temporally. The online discovery of specific trajectory patterns so-called "gatherings" has been tackled in [24]. However, their focus was rather on moving objects that share a stable and durable dense area.

Moving Objects Clustering The approaches proposed on [3], [4], [5], [25], [26] cluster moving object based on object spatial position at each timestamp and some of them incrementally maintain the clusters as time goes by. Some of them also consider that the objects move in a linear function model([3], [5], [25]) and predict when a moving object will leave or join a cluster. Instead of predict, CUTiS can observe the real displacement behavior of objects. Furthermore, [3], [25] do not apply density-based clustering, and [4] just consider update on moving object position as time goes by (no insertion and no deletion of moving objects).

Movement Pattern Discovery A very related topic to this study is to discover collective patterns among moving objects as flock, swarm, convoy, herds, gathering, among others [21]. The published approaches to find these patterns are very sensitive to specific parameters. Hence, these methods cannot be directly applicable for our problem since they do not report sub-trajectory clusters and their evolution. We believe that CUTiS enable to find some mobility patterns. For instance, flocks could be derived from micro-groups by a light post-processing since it is a subset of the later. The convoys are also similar to the density based clusters generated by our clustering algorithm. The representative trajectory is close notion to leadership.

VI. Conclusion and Future Work

In this paper, we consider the problem of discover and maintain sub-trajectory clusters at each time window, since each moving object updates its sub-trajectory from time to time, new moving objects appear and others disappear from the system. As each moving object evolves in time, also cluster evolves. We have presented an approach to track moving objects and incrementally maintain sub-trajectory clusters using trajectory data streams. CUTiS also captures the sub-trajectory cluster's evolution from time to time. Our experiments were conducted on real data sets, and it shows the efficiency and effectiveness of CUTiS compared to our competitors. In the future, we are going to extend our method to find others mobility patterns.

Acknowledgement. Research supported by CNPq in Brazil and performed while the first author was visiting David Laboratory in UVSQ, France.

References

[1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. Incremental clustering for mining in a data warehousing environment. In *VLDB*, volume 98, pages 323–333, 1998.
[2] Yifan Li, Jiawei Han, and Jiong Yang. Clustering moving objects. In *SIGKDD*, pages 617–622, 2004.
[3] Christian S. Jensen, D. Lin, and Beng-Chin Ooi. Continuous clustering of moving objects. *IEEE TKDE*, 19, 2007.
[4] Lu-An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Chih-Chieh Hung, and Wen-Chih Peng. On discovery of traveling companions from streaming trajectories. In *ICDE*, pages 186–197, 2012.
[5] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. Effective online group discovery in trajectory databases. *TKDE*, 25:2752–2766, 2013.
[6] Mirco Nanni and Dino Pedreschi. Time-focused clustering of trajectories of moving objects. *JIIS*, 27:267–289, 2006.
[7] Nikos Pelekis, Ioannis Kopanakis, Evangelos E Kotsifakos, Elias Frentzos, and Yannis Theodoridis. Clustering uncertain trajectories. *KAIS*, 28:117–147, 2011.
[8] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.
[9] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *VLDB*, 1:1081–1094, 2008.
[10] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.
[11] Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez. Traffic density-based discovery of hot routes in road networks. In *SSTD*, pages 441–459. 2007.
[12] Binh Han, Ling Liu, and Edward Omiecinski. Neat: Road network aware trajectory clustering. In *ICDCS*, pages 142–151, 2012.
[13] Ahmed Kharrat, Iulian Sandu Popa, Karine Zeitouni, and Sami Faiz. Clustering algorithm for network constraint trajectories. In *Headway in Spatial Data Handling*, pages 631–647. Springer, 2008.
[14] Martin Ester, Hans-Peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.
[15] Ticiana Coelho da Silva, Karine Zeitouni, José A F de Macêdo, and Marco A Casanova. On-line mobility pattern discovering using trajectory data. In *EDBT*, pages 682–683, 2016.
[16] Elias Frentzos, Kostas Gratsias, Nikos Pelekis, and Yannis Theodoridis. Algorithms for nearest neighbor search on moving object trajectories. *Geoinformatica*, 11:159–193, 2007.
[17] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.
[18] Costas Panagiotakis, Nikos Pelekis, Ioannis Kopanakis, Emmanuel Ramasso, and Yannis Theodoridis. Segmentation and sampling of moving object trajectories based on representativeness. *TKDE*, pages 1328–1343, 2012.
[19] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, volume 6, pages 328–339, 2006.
[20] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *SIGKDD*, pages 133–142, 2007.
[21] Yu Zheng. Trajectory data mining: an overview. *TIST*, page 29, 2015.
[22] Zhenhui Li, Jae-Gil Lee, Xiaolei Li, and Jiawei Han. Incremental clustering for trajectories. In *DASFAA*, pages 32–46, 2010.
[23] Yu Zheng. T-drive trajectory data sample. August 2011.
[24] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. On discovery of gathering patterns from trajectories. In *ICDE*, pages 242–253, 2013.
[25] Yifan Li, Jiawei Han, and Jiong Yang. Clustering moving objects. In *SIGKDD*, pages 617–622, 2004.
[26] Ticiana Coelho da Silva, José de Macêdo, and Marco A Casanova. Discovering frequent mobility patterns on moving object data. In *SIGSPATIAL on MobiGIS*, pages 60–67, 2014.