Discovering Evolving Moving Object Groups from Massive-Scale Trajectory Streams

Ruoshan Lan[1], Yanwei Yu[1]*, Lei Cao[2], Peng Song[1], Yingjie Wang[1]
[1]School of Computer and Control Engineering, Yantai University, Shandong 264005, China
lanruoshan@gmail.com, {yuyanwei, pengsong, wangyingjie}@ytu.edu.cn
[2]CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA
lcao@csail.mit.edu

***Abstract*—The increasing pervasiveness of object tracking technologies leads to huge volumes of spatio-temporal data collected in the form of trajectory streams. The discovery of useful group patterns from moving objects' movement behaviors in trajectory streams is critical for real time applications ranging from transportation management to military surveillance. In this work we propose a novel type of group pattern, called evolving group, which models the unusual group events of moving objects that travel together within density connected clusters in evolving streaming trajectories. Our theoretical analysis and empirical study on the Beijing Taxi data demonstrate its effectiveness in capturing development, evolution and trend of group events of moving objects in streaming context. Furthermore, we propose a discovery framework that efficiently supports online detection of evolving groups over massive-scale trajectory streams using sliding window. It contains three phases along with a set of novel optimization techniques designed to minimize the computation costs. Our comprehensive empirical study demonstrates that our discovery framework is effective and efficient on real-world high volume trajectory streams.**

***Index Terms*—Moving Objects, pattern mining, evolving group pattern, trajectory streams**

I. Introduction

In recent years, location tracking technologies have been broadly utilized in a variety of applications ranging from traffic management to mobile social networks that have generated huge volumes of trajectory data from moving objects including people, vehicles and animals, etc. Such trajectory data can be utilized for different purposes, for instance travel-route prediction, friend recommendation, anomaly detection, and traffic control [1]. In this work, we focus on effectively detecting a particular type of movement pattern called *evolving group* from massive-scale moving object trajectory streams.

Evolving group pattern is a special type of group pattern that models the behavior of the moving objects that travel together. Techniques have been proposed in the literature to detect group patterns such as *flock* [8], *convoy* [9], *swarm* [10] and *gathering* [15] [16]. Among these techniques, the *gathering* pattern is the closest to our evolving group pattern. Gathering models the group event in the streaming trajectory data that involves congregation of moving objects. A gathering is defined as a sequence of density-based clusters in a period of at least k_c consecutive time points in which adjacent clusters are within a given distance range d. Moreover, each cluster is required to contain at least m_p dedicated moving objects (so-called *participators*) which are involved in at least k_p clusters, although not necessarily to be consecutive. However, the *gathering* pattern requires at least k_c consecutive time points. This might result in the loss of interesting sequence of moving object clusters.

Fig. 1 illustrates an example of *gathering* patterns. Let m_c = 3 (minimal # of objects for a cluster), k_c = 3 (minimal duration of a gathering), k_p = 2 (minimal # of participated clusters for a participator) and m_p = 3 (minimal # of participators). Suppose cluster c_3 is far away from c_2 and c_4 is far away from c_1, namely, $distance(c_2, c_3) > d$ and $distance(c_1, c_4) > d$. In this example, the two sequences of clusters $\langle c_1, c_3, c_5\rangle$ and $\langle c_2, c_4, c_5\rangle$ form two gathering candidates from t_1 to t_3. In this case, only $\langle c_1, c_3, c_5\rangle$ is a gathering pattern with participator set $\{o_1, o_2, o_3, o_4\}$, because it contains at least three participators at all the time points, whereas $\langle c_2, c_4, c_5\rangle$ (participator set $\{o_3, o_4, o_5, o_6\}$) only contains two participator $\{o_3, o_4\}$ in c_5. Similarly, from t_5 to t_7, there are two gathering candidates $\langle c_6, c_7, c_9\rangle$ and $\langle c_6, c_8, c_9\rangle$. However, only $\langle c_6, c_7, c_9\rangle$ forms a gathering since each cluster includes at least three participators.

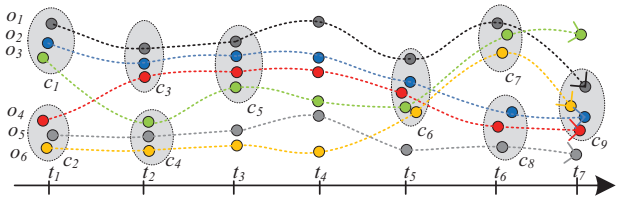


Figure 1: Comparison of gathering and evolving group

Using *gathering* pattern, we can get two independent gatherings from this example, because gathering requires the participators of a group to move together in consecutive clusters during its lifetime. However, understanding the trend and evolution of group pattern in streaming environments is considered to be more useful and helpful than simply extracting the group patterns. For example, the causal interactions of discovered groups along time can reveal the inherent relationships of groups of moving objects. In the above example we find that there are four participators $\{o_1, o_2, o_3, o_4\}$ in the

Ruoshan Lan and Yanwei Yu are co-first authors. Corresponding author: Yanwei Yu, yuyanwei@ytu.edu.cn.

2375-0324/17 $31.00 © 2017 IEEE
DOI 10.1109/MDM.2017.42

gathering $\langle c_1, c_3, c_5\rangle$ and five participators $\{o_1, o_2, o_3, o_4, o_6\}$ in $\langle c_6, c_7, c_9\rangle$. It is obvious that $\langle c_6, c_7, c_9\rangle$ is developed from $\langle c_1, c_3, c_5\rangle$ and grows larger. However, the *gathering* pattern is not suitable for modeling the variational groups in which memberships evolve gradually in streaming environment because of the strict consecutive moving together requirement. Although an incremental solution is proposed in [16] to support online discovery of *gatherings* pattern form the most recent arrived data in the trajectory stream, it does not take the evolving behavior of trajectories into account.

To overcome the above shortcomings of gathering pattern, we define an *evolving* group as a dense group of objects that share common behaviors in *most of the time* and change gradually over time. Unlike *gathering* pattern, our evolving pattern does not require the participators to strictly moving as a group lifelong. This successfully models the scenarios that gathering cannot discover. For example, the crowding might become "eased somewhat" at some time points in a traffic jam. Hence no enough dense cluster is formed occasionally. But they might again fall into a congestion soon at next crossroad. This type of group behavior can be recognized by our evolving pattern.

Another characteristic of evolving group is that it does not require coherent membership over streams. Namely, the members could change gradually over time. *Gathering* pattern allows members to join and leave group at any time and captures the gradual change of *participator (core members)* in streams, i.e., the evolution of participators. Furthermore, our definition of evolving group is suitable for capturing the group patterns from high-volume evolving streaming data.

More specially, our *evolving group* captures the evolving behavior of core group members at aggregation level over trajectory streams by introducing the notions of "crowd" and "gathering" **in sliding window**. The "gathering" is a special "crowd" that contains at least m_p participators in each cluster. Then we define the "group" as the set of participators in the "gathering" of a window to model the *core members*. We further define the *evolving group* to refer to a sequence of evolved "groups" that share most members (e.g. m_g) in two adjacent time windows to capture the evolution of *core members*.

In Fig. 1, let window size be 4, m_g = 3. Other parameters are the same with *gathering* pattern. There is a "gathering" $\langle c_1, c_3, c_5\rangle$ along with a "group" g_1=$\{o_1, o_2, o_3, o_4\}$ in window $W_{t_1}^{t_4}$. Suppose $distance(c_5, c_6) < d$, $\langle c_3, c_5, c_6\rangle$ also forms a "gathering" in $W_{t_2}^{t_5}$ and evolves from $\langle c_1, c_3, c_5\rangle$ with same "group" g_2=$\{o_1, o_2, o_3, o_4\}$. After the time t_6 arrives, the window slides from $W_{t_2}^{t_5}$ to $W_{t_3}^{t_6}$. Two "crowds" $\langle c_5, c_6, c_7\rangle$ and $\langle c_5, c_6, c_8\rangle$ are discovered, whereas only $\langle c_5, c_6, c_7\rangle$ is a "gathering" that corresponds to the "group" g_3=$\{o_1, o_2, o_3, o_4, o_6\}$, which evolves from g_2 and grows larger. In new window $W_{t_4}^{t_7}$, there is also only one "gathering" $\langle c_6, c_7, c_9\rangle$ that satisfies m_p participators at each cluster. The corresponding "group" g_4=$\{o_1, o_2, o_3, o_4, o_6\}$ evolves and keeps from last window. Therefore, an *evolving group* $\langle g_1, g_2, g_3, g_4\rangle$ reveals the evolution and trend of the group behavior shown in the example.

The second problem we must deal with is the *design of a solution that can efficiently discover evolving groups from large-scale streaming trajectories.* Many applications require the real time monitoring and tracking of moving objects to discover the groups as soon as possible for supporting online decision-making. Hence the algorithm should simultaneously report the results while receiving and processing the massive-scale trajectory streams. Apparently we can not simply apply or extend existing mining algorithms of group patterns to support the discovery of our new proposed evolving group patterns. In this paper, we propose an online discovery framework of the *evolving group*, which contains three phases: 1) discover all closed "crowds" in the current window, 2) detect all closed "gatherings" with their corresponding "groups" from the discovered crowds, and 3) update the evolving groups.

The main contributions of the paper are as follow:

- 1) We propose the new concept of evolving group pattern that captures the interesting group patterns over streaming trajectories that cannot be captured by the current group pattern detection techniques.
- 2) We propose an efficient evolving group discovery framework that efficiently discovers the evolving groups from large scale trajectory streams with real time responsiveness.
- 3) Extensive empirical study is conducted on the real traffic data (*Taxi* dataset) to evaluate the effectiveness and efficiency of the proposed framework. Our results offer insight into the effectiveness of the proposed framework in capturing evolving group events over trajectory streams.

II. Related Work

Most of the related work on group pattern mining has been discussed in Section 1. In this section, we briefly review some other representative works that are also related to our problem.

Trajectory clustering. Gaffney et al. [2] first propose the fundamental principles of clustering trajectories based on probabilistic modelling. They consider the trajectory as a whole and represent a set of trajectories using a mixture of regression models. Lee et al. [3] propose a partition and group framework to discover common sub-trajectory clusters in static trajectory databases. Jensen et al. [4] present a scheme that is capable of incrementally clustering moving objects by employing object dissimilarity and clustering features to improve the clustering effectiveness. More recently, Li et al. [5] further propose an incremental trajectory clustering framework that contains online micro-cluster maintenance and offline macro-cluster creation for incremental trajectory databases.

Co-movement pattern discovery in static databases. One of the earliest works on co-movement discovery is introduced by Laube and Imfeld [6] and further $flock$ [8] [11], $convoy$ [9], $swarm$ [10] and $gathering$ [15] are studied by others. Kalnis et al. [7] propose one similar notion of moving cluster, which is a set of objects when they can be clustered at consecutive time points, and the portion of common objects in any two consecutive clusters is not below

a predefined threshold. Another similar notion, *moving group pattern* [12], relies on disk-based clustering to mine a set of objects that are within a distance threshold from one another for a minimum duration. Similar with $swarm$, *moving group pattern* also permits members of pattern to travel together for a number of nonconsecutive timestamps by a weight threshold. A recent study by Li et al. [13] proposes the notion of $group$ that uses density connectedness for clustering trajectories without relying on sampling points. Namely, the group pattern simultaneously satisfies sampling independence, density connectedness, trajectory approximation and online processing.

Group pattern discovery in trajectory streams. Tang et al [14] recently propose the problem of discovering travelling companions in the context of streaming trajectories. The notion of travelling companion is as essential as $convoy$ [9]. However, they work on incremental algorithm of pattern discovery when the trajectories of users arrive in form of data streams. Vieira et al. [11] present the online flock discovery solution in polynomial time by identifying a discrete number of locations to place the center of the flock disk inside the spatial universe. Aung and Tan [17] propose the notion of evolving convoys to better understand the states of convoys. Specifically, an evolving convoy contains both dynamic members and persisted members. Zheng et al. propose the online *gathering pattern* [15] discovery over trajectories in an incremental manner in [16], which can capture congregations of moving individuals incrementally from durable and stable area with high density in trajectories. However, our goal is totally different from these work. These studies try to find object clusters for consecutive duration of time points over trajectories, while our work attempts to discover evolving groups of dynamic core objects in most recent trajectory streams.

III. Problem Definition

In this section, we introduce the definitions of all concepts used throughout the paper, and formally state the focal problem to be solved. The list of major symbols and notations in this paper is summarized in Table I.

We adopt the notion of density-based clustering [18] to define the snapshot cluster. C_{t_i}=$\{c_{t_i}^1, c_{t_i}^2, \ldots, c_{t_i}^m\}$ is the collection of snapshot clusters at timestamp t_i.

In this work, we use the periodic sliding window semantics to define the sub-stream of an infinite trajectory data stream. Each window W has a starting time $W.s$ and an ending time $W.e = W.s+w-1$, where w is a predefined window size. The window whose $W.e$ equals to the current timebin is called the current window, denoted as W_c. We also use $W_{t_i}^{t_j}$ to denote the window whose $W.s$=t_i and $W.e$=t_j. Now we first introduce our "crowd" concept in sliding window as following Def. 1.

***Definition** 1:* **(Crowd)** Given a trajectory stream in the sliding window W, a support threshold m_c, a distance threshold d, and a timestamp threshold k_c, a crowd Cr is a sequence of clusters at non-consecutive timestamps, i.e., Cr =$\langle c_{t_a}, c_{t_b}, \ldots, c_{t_k}\rangle$ ($W.s \leq t_a < t_b < \cdots < t_k \leq W.e$), which satisfies the following requirements:

Table I: Table of notations

Notation	Definition
o_i	a moving object
O_{DB}	the set of all moving objects
t_i	timestamp at the i^{th} time point
c_{t_i}	a cluster at timestamp t_i
Cr	a crowd in a window
k_c	the timestamp threshold of a crowd
m_c	the support threshold of a crowd
d	the distance threshold in crowd
k_p	the cluster support threshold of a participator
m_p	the participator count threshold of a gathering
m_g	The support threshold for evolving group
W, w	a sliding window and its window size
W_s,W_e	The starting and ending time of a window W
Ga	a gathering in a window
Gr	a group in a window
$CanSet$	the set of closed crowd candidate in a window
$c_{t_i}.st$	the status flag of ending cluster c_{t_i}
$o_i.cnt$	the # of clusters where o appears in a crowd

(1) The number of clusters in Cr, i.e the number of timestamps, is not less than k_c.
(2) There should be at least m_c objects in each cluster of Cr.
(3) The distance between any adjacent pair of clusters in Cr is not greater than $d*\Delta t$, where Δt is time difference between the pair of clusters.

Intuitively, a crowd is bounded in a sliding window, so k_c should be less than w. If k_c is equal to w, our "crowd" degenerates to the $crowd$ in [15]. We also use the Hausdorff distance [19] to measure the distance between two clusters.

Moreover, a crowd Cr is said to be closed iff there is no superset of Cr which is a crowd in the current window. Our goal is to find the closed crowds to avoid discovering redundant crowds. Essentially the concept of crowd in sliding window can capture dense group of object clusters in most recent time. Unlike $crowd$ in [15], we don't require that clusters in a crowd are consecutive, i.e., our "crowd" has a relaxed time restriction. Furthermore, the adjacent clusters (here adjacent does not mean adjacent in time) should satisfy the condition that their distance should be less than $d * \Delta t$, namely, we enlarge the distance threshold by being proportional to time difference to connect the clusters at the non-consecutive timestamps reasonably. Hausdorff distance obeys metric properties [19], namely, Hausdorff distance has the properties of identity, symmetry, and triangle inequality. This guarantees the subset of Cr is also a crowd if the length is not less than k_c, meaning that crowd also satisfies the downward closure property. Before defining the evolve group, we redefine the notions of participator and gathering in sliding window first.

***Definition** 2:* **(Participator)** Given a crowd Cr in the current window W, a cluster support threshold k_p, an object o is called a participator of Cr iff it appears in at least k_p

clusters of Cr.

Definition 3: **(Gathering)** Given a crowd Cr in the current window W, participator count threshold m_p, Cr is called a gathering iff each cluster of Cr includes at least m_p participators.

By Def. 2, a participator needn't stay in each cluster of the crowd. As long as an object occurs in the crowd for long enough timestamps, it is regarded as a participator. A gathering also needn't require a participator to occur at each time point in the window. Even if some clusters of a crowd Cr do not include enough participators, its sub-crowd may still be a gathering if the sub-crowd satisfies the constraint of m_p participators in each cluster. If a crowd Cr is a gathering in the current window W_c and there is no super-crowd $Cr' \supset Cr$ that is a gathering, then Cr is a closed gathering in W_c. Therefore, in each window we still need to detect the closed gatherings by exploring sub-crowds space although we aim to find the closed crowds to reduce redundant crowds.

Definition 4: **(Group)** Given a gathering Cr in the current window W, the set of all participators in Cr is called a group that corresponds to Cr in W.

We can also refer the concept of group to the *core members* of the gathering in current window. Next we will introduce the concept of evolving group that is exactly the problem which this paper studies.

Definition 5: **(Evolving Group)** Given a group Gr_1 in W_i, a group Gr_2 in W_{i+1}, a fraction support threshold m_g, Gr_2 is evolved from Gr_1 in streaming environments iff $|Gr_1 \cap Gr_2| \geq m_g * Min(|Gr_1|, |Gr_2|)$, denoted as $\langle Gr_1, Gr_2 \rangle$. Given a lifetime support threshold k_g, an evolving group is a chain of groups during at least k_g consecutive windows, i.e. $\langle Gr_{t_a}, Gr_{t_{a+1}}, \ldots, Gr_{t_b} \rangle$, where $Gr_{t_{i+1}}$ is evolved from Gr_{t_i} $(i = a, a+1, \ldots, b-1)$ and $|b-a+1| \geq k_g$.

IV. Discovering Evolving Groups

In this section, we now present our framework for discovering all evolving groups over trajectories in streaming window environment. Basically, our framework includes three phases: online crowd discovery, online group detection, and evolving group updating.

A. Online Crowd Discovery in Sliding Window

We first introduce the incremental algorithm, named **Incre**, which discovers the closed crowds in sliding window environment. It first utilizes the obtained clusters (by DBSCAN) to combine the sequences of clusters at new timestamp. Then the sequences of clusters are used as candidates to validate if they are closed. By leveraging overlap of adjacent windows in sliding window, our incremental algorithm successfully avoids the redundant crowd searching at previously timestamp. Using an example-driven approach, we now describe how the Incre algorithm detects the closed crowds.

Example 1: We use the example in Table II to illustrate the discovery of closed crowds in sliding window. To keep its simplicity, we assume the two clusters in the same row or adjacent rows are close to each other, i.e., their Hausdorff distance is not greater than $d * \Delta t$.

Table II: Example 1. snapshot clusters in sliding window

t_1	t_2	t_3	t_4	t_5
	c_2^1			c_5^2
		c_3^1		
c_1^1			c_4^1	c_5^1
c_1^2				

Table III: Illustration of closed crowd discovery

time	$CanSet$	$endclu$	$status$	$CloCr$
t_1	$\langle c_1^1 \rangle$;$\langle c_1^2 \rangle$	c_1^1;c_1^2		
t_2	$\langle c_2^1 \rangle$;$\langle c_1^1 \rangle$;$\langle c_1^2 \rangle$	c_2^1;c_1^1; c_1^2	$c_1^1.st$=$unmth$ $c_1^2.st$=$unmth$	
t_3	$\langle c_2^1, c_3^1 \rangle$; $\langle c_1^1, c_3^1 \rangle$; $\langle c_2^1 \rangle$; $\langle c_1^1 \rangle$; $\langle c_1^2 \rangle$	c_3^1;c_2^1; c_1^1;c_1^2	$c_2^1.st$=$match$ $c_1^1.st$=$match$ $c_1^2.st$=$unmth$	
t_4	$\langle c_2^1, c_3^1, c_4^1 \rangle$; $\langle c_1^1, c_3^1, c_4^1 \rangle$; $\langle c_1^2, c_4^1 \rangle$; $\langle c_2^1, c_3^1 \rangle$; $\langle c_1^1, c_3^1 \rangle$; $\langle c_2^1 \rangle$; $\langle c_1^1 \rangle$; $\langle c_1^2 \rangle$	c_4^1;c_3^1; c_2^1;c_1^1; c_1^2	$c_3^1.st$=$match$ $c_2^1.st$=$match$ $c_1^1.st$=$match$ $c_1^2.st$=$match$	$\langle c_2^1, c_3^1, c_4^1 \rangle$; $\langle c_1^1, c_3^1, c_4^1 \rangle$
~~t_{I}~~	$\langle c_2^1, c_3^1, c_4^1 \rangle$; ~~$\langle c_3^1, c_4^1 \rangle$;~~ ~~$\langle c_4^1 \rangle$;~~ $\langle c_2^1, c_3^1 \rangle$; ~~$\langle c_3^1 \rangle$;~~ $\langle c_2^1 \rangle$	c_4^1;c_3^1; c_2^1		
t_5	$\langle c_2^1, c_3^1, c_4^1, c_5^1 \rangle$; $\langle c_2^1, c_3^1, c_5^2 \rangle$; $\langle c_2^1, c_3^1, c_4^1 \rangle$; ~~$\langle c_2^1, c_3^1 \rangle$;~~ ~~$\langle c_2^1 \rangle$~~	c_5^1;c_5^2; c_4^1;c_3^1; c_2^1	$c_4^1.st$=$match$ $c_3^1.st$=$match$ $c_2^1.st$=$match$ $c_4^1.st$=$unmth$ $c_3^1.st$=$match$ $c_2^1.st$=$match$	$\langle c_2^1, c_3^1, c_4^1, c_5^1 \rangle$; $\langle c_2^1, c_3^1, c_5^2 \rangle$

We now first introduce the data structures used in our algorithm. We use $CanSet$ to store all closed crowd candidates in the current window W_c. To support closure check, each candidate Cr maintains a flag $Cr.endclu$ to indicate the ending cluster of Cr in W_c. For each ending cluster, we need to validate whether there exists a new cluster which can be appended to Cr in next window. Therefore, a status flag of $endclu$, denoted as $endclu.st$, is maintained for each ending cluster, which is set to $uncheck$ initially at the beginning of each new window. If there exists a new cluster c_{new} that be appended to the ending cluster $endclu$, then $endclu.st$=$match$, otherwise $endclu.st$ is labelled to unmatch (abbr. $unmth$). We use $CloCr$ to denote the closed crowds at each window. In particular, it is easy to observe that a closed crowd in the current window can be directly checked in the next window according to the following lemma.

Lemma 1: Given a closed crowd $Cr = \langle c_{t_{i+a}}, \ldots, c_{t_{j-b}} \rangle$ $(a \geq 0, b \geq 0)$ in window $W_{t_i}^{t_j}$, if $\exists c_{t_{j+1}}$ in window $W_{t_i}^{t_{j+1}}$

such that $Cr_{new} = \langle c_{t_{i+a}}, \ldots, c_{t_{j-b}}, c_{t_{j+1}} \rangle$ is a new crowd, then Cr_{new} is a closed crowd in $W_{t_i}^{t_{j+1}}$.

By Lemma 1, we can discover the closed crowds in the first window by incrementally validating the clusters at each new timestamp. The intermediate status of crowds at each timestamp is shown in Table III. Suppose $w = 4, k_c = 3$. There are two clusters c_1^1 and c_1^2 at time t_1. Hence two candidates $\langle c_1^1 \rangle$ and $\langle c_1^2 \rangle$ are stored in $CanSet$, and the ending clusters are inserted into $endclu$. After t_2 arrives, we get a cluster c_2^1, however, c_2^1 is very far away from the clusters at t_1. Therefore, no crowd candidate is updated, and the statuses of c_1^1 and c_1^2 are labelled to $unmth$. At t_3, there is a cluster c_3^1 being close to c_2^1 and c_1^1, thus $\langle c_2^1, c_3^1 \rangle$ and $\langle c_1^1, c_3^1 \rangle$ are inserted into $CanSet$. $c_2^1.st$ and $c_1^1.st$ are set to $match$, while $c_1^2.st$ is set to $unmth$. However, the candidates of last timestamp are still stored in $CanSet$ for validating potential crowd candidates in next timestamp. After t_4 arrives, we find that c_4^1 is close with c_3^1, thus $\langle c_2^1, c_3^1, c_4^1 \rangle$ and $\langle c_1^1, c_3^1, c_4^1 \rangle$ are generated as closed crowd candidates in window $W_{t_1}^{t_4}$. Hence $c_3^1.st$=$match$. Moreover, we also observe that $\langle c_1^1, c_4^1 \rangle$ is already included in $\langle c_1^1, c_3^1, c_4^1 \rangle$ although c_1^1 is also close to c_4^1 w.r.t. $d * \Delta t$. $\langle c_2^1, c_4^1 \rangle$ is also included in $\langle c_2^1, c_3^1, c_4^1 \rangle$ even assuming that c_2^1 was close to c_4^1. Namely, any other candidates in $CanSet$ ending with c_2^1 or c_1^1 must not form a closed crowd in the current window. Therefore, we set $c_2^1.st$ and $c_1^1.st$ to $match$ directly instead of validating the candidates with c_4^1 again. We can further deduce the following corollary.

***Corollary** 1:* Given a crowd candidate $Cr = \langle c_{t_{i+a}}, \ldots, c_{t_{j-b}} \rangle$ $(a \geq 0, b \geq 1)$ in window $W_{t_i}^{t_j}$, if $c_{t_{j-b}}$ is already included in a closed crowd candidate in $W_{t_i}^{t_{j+1}}$, then $Cr + \langle c_{t_{j+1}} \rangle$ must not form a closed crowd in $W_{t_i}^{t_{j+1}}$.

Then we can quickly detect all closed crowds in $W_{t_1}^{t_4}$. Namely, the candidates that end with clusters at t_4 or clusters whose status are $unmth$, and have number of clusters not less than $k_c(3)$. So $\langle c_2^1, c_3^1, c_4^1 \rangle$ and $\langle c_1^1, c_3^1, c_4^1 \rangle$ are reported as closed crowds in final.

After t_5 arrives, the window slides from $W_{t_1}^{t_4}$ to $W_{t_2}^{t_5}$. Since time t_1 has expired, all clusters of t_1 are removed from $CanSet$ and $endclu$, as shown in Table III. But we can see that some sub-crowds incur due to the remove of t_1. *Which sub-crowds should be removed? And which sub-crowds should be reserved?* We observe that the sub-crowd that shares the ending clusters with its super-crowd should be removed from $CanSet$ because it must be included in a close crowd candidate in next window. On the other hand, the sub-crowd that does not include the ending cluster of its super-crowd should be reserved since it may form a close crowd candidate with new cluster in next window. Therefore, $\langle c_2^1, c_3^1 \rangle$ is reserved, while other sub-crowd candidates are removed from $CanSet$. At t_5, there are two clusters c_5^1 and c_5^2. For validating these two clusters, we need to store two copes of $CanSet$, $endclus$ and their status. For c_5^1 detection, we generate a new crowd candidate $\langle c_2^1, c_3^1, c_4^1, c_5^1 \rangle$ since c_5^1 is near c_4^1, and the statuses of c_4^1, c_3^1 and c_2^1 are labelled to $match$. For c_5^2 detection, since c_5^2 is far from c_4^1 but is close to c_3^1, we generate a new crowd candidate $\langle c_2^1, c_3^1, c_5^2 \rangle$ to insert into $CanSet$, and set $c_4^1.st$ to $unmth$, $c_3^1.st$ and $c_2^1.st$ to $match$.

Again, we can output the closed crowds from $CanSet$, $\langle c_2^1, c_3^1, c_4^1, c_5^1 \rangle$ and $\langle c_2^1, c_3^1, c_5^2 \rangle$, namely, candidates ending with clusters at t_5 or clusters whose st is $unmth$ at all copies.

1) Optimization strategies.: Next, we present two optimization strategies to minimize computation cost. It is easy to see that the number of $CanSet$ has a great impact on cost of our algorithm. We first propose the Lemma 2 to reduce the unnecessary maintained candidates.

***Lemma** 2:* Given a crowd candidate $Cr = \langle c_{t_{i+a}}, \ldots, c_{t_{j-b}} \rangle$ $(a \geq 0, b \geq 0)$ in window $W_{t_i}^{t_j}$, and support threshold k_c, if $b > j - i + 1 - k_c$, then Cr is not a crowd or part of a crowd from $W_{t_i}^{t_j}$ till to expiration of Cr.

Lemma 2 is intuitive. We omit the proof due to the space limitations. Using the Lemma 2, the optimized $CanSet$ in window $W_{t_2}^{t_5}$ is shown as last row in Table III.

Furthermore, we also observe that the order of candidates in $CanSet$ and corresponding ending clusters can improve the efficiency of our algorithm by pruning the unnecessary verification for new clusters in the new window.

***Lemma** 3:* Given a candidate $Cr = \langle c_{t_{i+a}}, \ldots, c_{t_{j-b}} \rangle$ $(a \geq 0, b \geq 0)$ in window $W_{t_i}^{t_j}$, and a cluster $c_{t_{j+1}}$, if $d_H(c_{t_{j-b}}, c_{t_{j+1}}) \leq d * (b + 1)$, then no need to validate the candidates that end with any c in Cr-$\langle c_{t_{j-b}} \rangle$ with $c_{t_{j+1}}$ again.

Lemma 3 can be easily proved by Corollary 1. Based on the observation, we sort $CanSet$ in the last time first order by ending clusters, and also store corresponding ending clusters in last time first order in $endclu$ copy at each new timestamp. For validating a new cluster c_{new}, we traverse the candidates with the latest time ending cluster, if c_{new} is appended into a candidate Cr, all clusters in Cr are labelled to $match$ if they are in $endclu$ copy, i.e. we will skip the candidates that end with the clusters in next traversal on $CanSet$ for c_{new}.

2) Cluster pruning strategy.: Indexing clusters with R-tree or grid can improve the efficiency of discovery algorithm. However the index methods suffer from two major drawbacks. First, indexing does not work very well in dynamic streaming environments in which the index has to be continuously rebuilt when streaming data evolves. Second, it is also very expensive for mapping the clusters that include lots of objects into index. Here, we propose a pruning cluster strategy to reduce the number of Hausdorff distance computation.

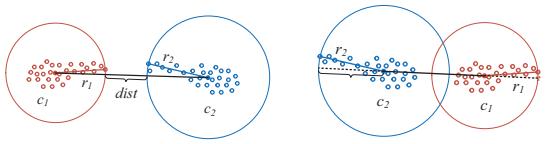
To start, we use the mean center of cluster m_i and maximum radius r_i to represent the cluster c_i. r_i is the distance from the center m_i to the farthest point in c_i. Intuitively, we get following two pruning rules.

***Rule** 1:* (Long-distance Pruning) Consider two clusters c_i and c_j, their mean centers m_i and m_j, radius r_i and r_j, and time difference Δt, if the distance between m_i and m_j is greater than r_i+r_j+d*Δt, then the Hausdorff distance between c_i and c_j must be greater than $d * \Delta t$.

***Rule** 2:* (Short-distance Pruning) Consider two clusters c_i and c_j, their mean centers m_i and m_j, radius r_i and r_j, and time difference Δt, if the distance between m_i and m_j is not

greater than $d * \Delta t$-$Max(r_1, r_2)$, then the Hausdorff distance between c_i and c_j must not be greater than $d * \Delta t$.

We omit the proof of these rules due to the space limitations. As shown in Fig. 2(a), the smallest minimum distance from points in c_1 to c_2 must be larger than $dist$, similarly, the smallest minimum distance from points in c_2 to c_1 also must be larger than $dist$. Thus $d_H(c_1, c_2) > dist$. Therefore, if $|m_1 - m_2| > r_1 + r_2 + d * \Delta t$, namely, $dist > d * \Delta t$, then $d_H(c_1, c_c) > dist > d * \Delta t$. In Fig. 2(b), the maximum minimum distance from points in c_2 to c_1 must be less than r_2+$dist(m_1, m_2)$, similarly, the maximum minimum distance from points in c_1 to c_2 must be less than r_1+$dist(m_1, m_2)$. Hence, $d_H(c_1, c_2) < Max(r_1, r_2) + dist(m_1, m_2)$. If $dist(m_1, m_2) < d * \Delta t$-$Max(r_1, r_2)$, namely, $Max(r_1, r_2) + dist(m_1, m_2) < d * \Delta t$, then $d_H(c_1, c_2) < d * \Delta t$. By the pruning rules, we only first calculate the distance between the mean centers to determine whether to continue to compute the Hausdorff distance for validating the final results.



(a) Long-distance pruning (b) Short-distance pruning

Figure 2: Examples of pruning cluster

3) Incre algorithm.: Alg. 1 shows the Incre algorithm for discovering closed crowd in sliding windows. Incre first removes the information of t_{i-1} from $CanSet$ of last window. In particular, we also use Lemma 2 to remove the unnecessary maintained candidates (lines 4-5). $Cr.endclu.t$ denotes the timestamp of ending cluster of candidate Cr. For a new cluster c_{t_j}, we maintain a copy of $CanSet$, denoted as $c_{t_j}.CanSet$. Each candidate whose ending cluster is unchecked would be validated with c_{t_j}, as shown in lines 9-16. If c_{t_j} is close to $Cr.endclu$, a new candidate is generated and inserted into $CanSet$, and all ending clusters in Cr are labelled to $match$ (lines 11-14). $Cr.allendclu$ stands for all ending clusters in Cr. Finally, all length enough ($\geq k_c$) candidates that end with clusters at t_j or with $numth$ status in all copies are reported as closed crowds (lines 17-19). $Cr.endclu.allst = unmth$ denotes the statuses of $Cr.endclu$ are $unmath$ in $CanSet$ copies of all new clusters.

B. Online Group Extraction in sliding window

Next, we will discuss the algorithm to detect closed gathering and corresponding group on each obtained closed crowd in sliding window. We also first use an example to elaborate our process method. Then we will present the details of our algorithm on basis of the proposed principles.

***Example** 2:* Consider a closed crowd shown in Table IV, and let $w = 8$, $k_c = 6$, $k_p = 5$ and $m_p = 3$. There is a closed crowd $Cr_1 = \langle c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8} \rangle$ with 6

Algorithm 1 Incre algorithm for Crowd Discovery

Input: the current window $W_{t_i}^{t_j}$, C_{t_j}, k_c, m_c, d
Output: Closed crowds $CloCr$

```
1: CloCr ← ∅; CanSet ← W_{t_{i-1}}^{t_{j-1}}.CanSet;
2: for each Cr ∈ CanSet do
3:     Remove c_{t_{i-1}} from Cr; //delete clusters at time t_{i-1}
4:     if (|t_{j-1} − Cr.endclu.t| > j-i+1-k_c) then
5:         Remove Cr from CanSet;
6: CanSetCopy ← CanSet;
7: for each c_{t_j} ∈ C_{t_j} do
8:     c_{t_j}.CanSet ← CanSetCopy; // a copy of CanSet
9:     for each Cr ∈ c_{t_j}.CanSet do
10:        if (Cr.endclu.st == uncheck) then
11:            if (d_H(c_{t_j}, Cr.endclu) ≤ d * Δt) then
12:                Insert Cr + ⟨c_{t_j}⟩ into CanSet;
13:                Cr.endclu.st ← match;
14:                Cr.allendclu.st ← match;
15:            else
16:                Cr.endclu.st ← unmth;
17: for each Cr ∈ CanSet do
18:     if ((Cr.endclu.t == t_j | Cr.endclu.allst == unmth)
    &Cr.len ≥ k_c) then
19:         CloCr ← CloCr ∪ Cr;
```

Table IV: Example 2. a closed crowd in $W_{t_1}^{t_8}$ and $W_{t_2}^{t_9}$

c_{t_1}	c_{t_2}	c_{t_3}	c_{t_4}	c_{t_5}	t_6	c_{t_7}	c_{t_8}	c_{t_9}
o_1	o_1	o_1		o_1		o_1	o_1	
o_2	o_2		o_2	o_2		o_2	o_2	o_2
o_3	o_3		o_3	o_3		o_3	o_3	o_3
		o_4				o_4		
	o_5		o_5	o_5		o_5	o_5	o_5
o_6	o_6	o_6	o_6				o_6	

objects in window $W_{t_1}^{t_8}$ that also evolves to a closed crowd $Cr_2 = \langle c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8}, c_{t_9} \rangle$ in $W_{t_2}^{t_9}$.

We first verify if the crowd is a gathering in current window $W_{t_1}^{t_8}$. It is easy to see that the objects $\{o_1, o_2, o_3, o_5, o_6\}$ are participators w.r.t. threshold $k_p(5)$. But not every cluster contains $m_p(3)$ participators, e.g. c_{t_3} is considered as an invalid cluster since it only contains two participators. However, $\langle c_{t_1}, c_{t_2}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8} \rangle$ still is a crowd in this window by removing the invalid cluster, meaning that it still may be a gathering. Therefore, we need to further check the sub-crowd if it is a closed gathering. Again we get participators $\{o_1, o_2, o_3, o_5\}$ from the sub-crowd. Obviously, the set of participators is a subset of that of Cr_1. Now we introduce the following lemma by the observation.

***Lemma** 4:* Given a crowd Cr and its corresponding participator set O in window $W_{t_i}^{t_j}$, for any crowd $Cr' \subset Cr$, the corresponding participator set O' of Cr' is a subset of O.

Lemma 4 is intuitive. We omit the proof due to the space limitations.

In particular, we use $o.cnt$ to denote the number of clusters in which o appears in a crowd. For example, we get $\{o_1.cnt =$

$6, o_2.cnt = 6, o_3.cnt = 6, \overline{o_4.cnt = 2}, o_5.cnt = 5, o_6.cnt = 5\}$ in Cr_1, where the object with an overline denotes a non-participator because its cnt is less than $k_p(5)$. When re-checking the participators in $Cr_1 - \langle c_{t_3}\rangle$, we only verify if the participators appear in c_{t_3}. If yes, $o.cnt$ is reduced by 1, otherwise, $o.cnt$ stays the same. Therefore, we can fast find out new participator set $\{o_1.cnt = 5, o_2.cnt = 6, o_3.cnt = 6, o_5.cnt = 5, \overline{o_6.cnt = 4}\}$. Then we detect a closed gathering $\langle c_{t_1}, c_{t_2}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8}\rangle$ along with its group $\{o_1, o_2, o_3, o_5\}$, since each cluster contains at least $m_p(3)$ participators.

After t_9 arrives, $W^{t_8}_{t_1}$ slides to $W^{t_9}_{t_2}$. A new closed crowd Cr_2 in $W^{t_9}_{t_2}$ is generated based on Cr_1 in last step. First, we need to delete the information of t_1 from Cr_1. Similarly, we can directly use the above method to fast update participator set by removing invalid cluster c_{t_1} from Cr_1. Next, we consider that how to update new cluster c_{t_9} to obtained result using minimal computation. We here add the new cluster into $Cr_1 - \langle c_{t_1}\rangle$, namely, verify Cr_2 in basis of the obtained participator set. Therefore, we only need to check whether the objects appear in new cluster c_{t_9}. Thus, the participator set is updated to $\{o_1.cnt = 5, o_2.cnt = 6, o_3.cnt = 6, \overline{o_4.cnt = 2}, o_5.cnt = 6, \overline{o_6.cnt = 4}\}$.

Similarly, we next continue to verify the candidate using above removing method. Finally, $\langle c_{t_2}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8}, c_{t_9}\rangle$ is reported as a closed gathering with corresponding group $\{o_2, o_3, o_5\}$ in $W^{t_9}_{t_2}$.

Moreover, if c_{t_9} was an invalid cluster, we can see that the detection falls back to obtained result in last window, i.e. $W^{t_8}_{t_2}$. Namely, we only need to directly verify the obtained result in $W^{t_8}_{t_1}$ by removing t_1.

By the illustration of Example 2, we propose our **Verify with Removing and Adding** (VRA) algorithm to detect closed gatherings with corresponding groups in sliding window efficiently.

As shown in Alg. 2, Ga denotes a closed gathering, and Gr is its corresponding group. Par is the participator set with their cnts. If Cr is an emerging closed gathering, we get its participator set in clusters of Cr from scratch (denoted as $Participator(Cr)$). Otherwise, we can fast find out participator set of Cr from Par of last window by only verifying the objects in $c_{t_{i-1}}$ and c_{t_j} (lines 4-6). Lines 8-19 show the detection process of closed gathering applying a downward method. VRA first checks if each cluster in the crowd copy Cr' contains enough participators shown as lines 12-14. If not, VRA then updates the participators by removing invalid clusters (lines 9-11), and re-checks each clusters of the remainder again (lines 12-14) till the remainder is not a crowd or there is no invalid cluster. If there is no invalid cluster, Cr' and its current participators are reported as closed gathering and corresponding group. Actually, we only focus on the real participators whose $cnt \geq k_p$ instead of all objects in Cr in this detection process because the participators of a crowd must be from that of its super-crowd by Lemma 4. $Par'(k_p)$ denotes the set of real participators that satisfy k_p threshold, and $Par'(k_p, c)$ denotes the set of real participators in cluster c.

However, if c_{t_j} is an invalid cluster in the process, the detection falls back to obtained result in last window. As shown in lines 17-19, we further verify the obtained gathering of last window by removing the expired timestamp t_{i-1}.

Algorithm 2 Verify with Removing and Adding (VRA)

Input: closed crowd Cr in window $W^{t_j}_{t_i}$, closed crowd Cr_1 and Par in $W^{t_{j-1}}_{t_{i-1}}$, k_c, k_p, m_p
Output: gathering Ga, and corresponding group Gr

1: $Ga \leftarrow \emptyset$; $Gr \leftarrow \emptyset$; $C_{un} \leftarrow \emptyset$;
2: **if** Cr is an emerging closed crowd **then**
3: $\quad Par \leftarrow Participator(Cr)$;
4: **else**
5: $\quad Par \leftarrow Par - obj(c_{t_{i-1}})$; //delete information of t_{i-1}
6: $\quad Par \leftarrow Par + obj(c_{t_j})$; //add information of t_j
7: $Cr' \leftarrow Cr$; $Par' \leftarrow Par$;
8: **while** $(|Cr' - C_{un}| \geq k_c)$ **do**
9: $\quad$ **for** each $c \in C_{un}$ **do**
10: $\quad\quad Par' \leftarrow Par' - obj(c)$;
11: $\quad Cr' \leftarrow Cr' - C_{un}$; $C_{un} \leftarrow \emptyset$;
12: $\quad$ **for** each $c \in Cr'$ **do**
13: $\quad\quad$ **if** $(|Par'(k_p, c)| < m_p)$ **then**
14: $\quad\quad\quad C_{un} \leftarrow C_{un} \cup c$;
15: $\quad$ **if** $(C_{un} == \emptyset)$ **then**
16: $\quad\quad Ga \leftarrow (Cr')$; $Gr \leftarrow Par'(k_p)$; Break;
17: $\quad$ **else if** $(c_{t_j} \in C_{un}$ & $|Cr' - C_{un}| \geq k_c)$ **then**
18: $\quad\quad$ Fall back to Ga and Gr in $W^{t_{j-1}}_{t_{i-1}}$;
19: $\quad\quad$ Verify Ga-$\langle c_{t_{i-1}}\rangle$ on Gr; Break;

Lemma 5: Given a closed crowd Cr in the current window W_c, If Ga is reported as a gathering from Cr by VRA algorithm, then the gathering Ga is closed in W_c.

Proof 1: Lemma 5 can be proved easily. Suppose Ga is reported as a gathering by VRA algorithm. According to the downward flow of our VRA algorithm, Ga is the biggest sub-crowd of Cr that satisfies the condition that each cluster contains m_p participators. Namely, there is no super-crowd of Ga which is a gathering. Therefore, Ga is closed in W_c by closure property of gathering described in Sec. III.

C. Evolving Group Updating

In the third phase, we update the evolving groups by constructing the sequence chain of groups incrementally. For each group obtained in the current window, we first evaluate if it is evolved from a group of last window, namely, whether it shares most of core members with the groups detected in last window. If yes, we update the group into the evolving group. Therefore, we can find that each pair of adjacent groups in an evolving group has an evolutionary relationship. If the group does not share most of members with any group in last window, i.e., the group is an emerging gathering in the current window, a new evolving group that only contains the group is constructed. Moreover, If the last group in a chain (the group in last window) is not continued to be evolved in

current window, we say that the evolving group is a closed evolving group.

V. Experiment

All algorithms in the experiment are implemented in Java on CHAOS stream engine [20]. All tests run on a computer equipped with Inter Xeon E5-2660 CPU (2.2GHz), 16G memory, and Windows Server 2012 operating system.

Real Dataset. We use a real life trajectory dataset $Taxi$ in the experiment study. The dataset is from T-Drive project [21] collected by Microsoft Research Asia. In our experiments, we use a sample of the dataset that contains one week trajectories of 10,357 taxis in a period from February 2 to February 8 2008. We divide a day into four time periods, morning and evening peak time (7am to 10am and 4pm to 8pm), work time in morning and noon (10am to 1pm) and work time in afternoon (1pm to 4pm). We also interpolate the time domain into the granularity of minute on $Taxi$ dataset.

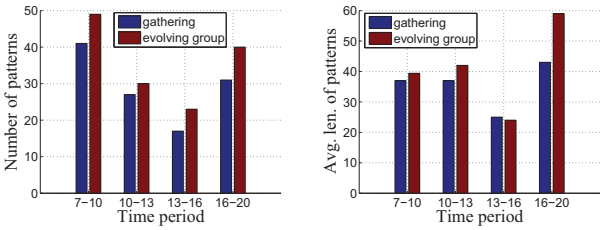
A. Effectiveness

First, we evaluate the effectiveness of our proposed evolving group in case study of traffic condition on $Taxi$ data compared against *gathering* pattern [15][16].

We obtain the snapshot clusters at each timestamp by setting $MinPts$=5 and Eps=300 meters. Fig. 3(a) shows the average number of patterns discovered by our proposed algorithm on a single day with the settings of w=6, k_c=5, m_c=8, k_p=4, m_p=5, m_g=0.7, and d=300 meters (i.e., a group of 5 or more core members travelling at least 5 time points in a 6 min sliding window). We select the closed evolving groups with $k_g \geq 9$ (i.e., an evolving group lasting for at least 9 consecutive windows). As comparison, we also search for the gathering patterns at the corresponding settings k_c=15, m_c=8, k_p=11, m_p=5 (i.e., a gathering of 5 or more participators travelling together for a period of at least 15 minutes).

In Fig. 3(a), we can find that the overall trend of evolving groups is consistent with the number of gathering patterns, which reflects the severe traffic congestion during the rush time on workday in Beijing. However, more evolving groups are captured in traffic streams compared with gathering pattern, especially during two peak times. This is because gathering only focuses on the serious traffic jams that last for a period of fixed consecutive time units, while our evolving group also tracks the short-lived aggregations of vehicles during non-consecutive time to monitor if they are becoming more serious or getting ease, except the long traffic congestions.

Next, we compare the average length of discovered patterns on a single day. We select the closed evolving groups whose length $k_g \geq 15$ to make number of evolving groups be equal to the number of gatherings. From Fig. 3(b), we can easily see that the average length of evolving group is larger than that of gathering at all time period, meaning that our group pattern can detect the crowding events earlier or track the trend of the events more time units. In particular, the average length of evolving group is much larger than that of gathering during evening peak time, reaching at 15 minutes gap. This may be because that our framework captures the vehicle group events before the serious traffic jams are formed between 16:40 and 5:10 by observing the discovered patterns.



(a) Avg. # of discovered patterns (b) Avg. length of a single pattern

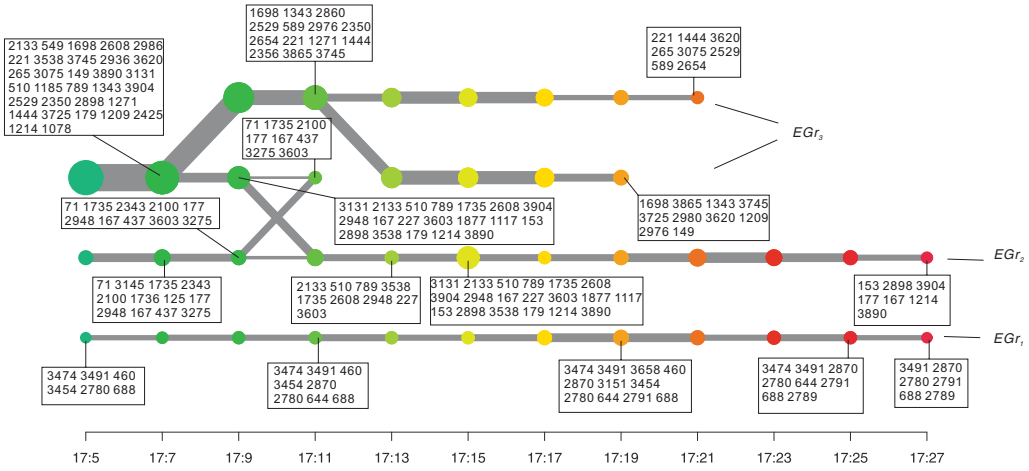
Figure 3: Effectiveness study

B. Case studies

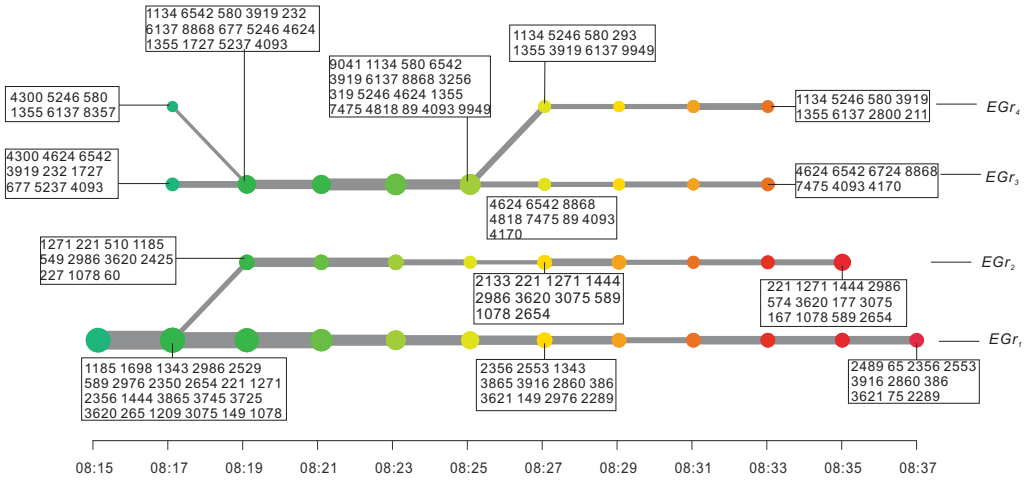
We employ a visualized method with two cases to demonstrate the effectiveness of our evolving group. As shown in Fig. 4, each circle represents a group in sliding window. For two connected groups by a solid line, the latter is evolved from the former, and the weight of line indicates the number of common members. We also use the size of circle to denote the number of objects in the group, and the color of circle to denote the life time of the evolving group, gradually changing from green to red. We also plot out the IDs of objects in the rectangle for some significant groups. It is easy to see the advantages of evolving groups from Fig. 4(a) and 4(b). Even if a gathering pattern has the same length as an evolving group, it can not contain such rich information as an evolving group reveals.

Case 1: Fig. 4(a) shows several evolving groups during the period of 17:05-17:27 on Feb 4. We simply mark out 3 representative evolving groups as shown in Fig. 4(a). Obviously, EGr_1 is independent of other evolving groups, however, it evolves continuously over time, for example, the evolving group grows to 11 core members at clock 17:19, while shrinks into 6 participators at 17:27. EGr_3 represents a serious traffic jam for 6 minutes at beginning of the case. However, the traffic jam gets alleviated by splitting into two smaller groups at 17:13, and then the two evolving groups end at about 17:19 and 17:21 respectively, meaning that transportation condition becomes much smoother. Another EGr_2 interacts with EGr_3 during [17:09, 17:11]. We observe that EGr_2 suddenly becomes much bigger at time 17:15, and then be much smaller at next window. This may be caused by an emergency, such as a traffic accident or an emergency repair.

Case 2: In Fig. 4(b), four evolving groups during 8:15-8:37 on Feb 4 are shown in this case. EGr_1 shows the development process of a continuous traffic congestion from very serious towards somewhat light. However, we can see that the participators at time 8:17, 8:27 and 8:37 also change significantly over time, but our model also captures the traffic jam using gradual evolution in sliding window. EGr_3 and EGr_4 share the first half part, which demonstrates the forming process of a serious traffic jam vividly. This can be revealed



(a) case 1



(b) case 2

Figure 4: Case studies on effectiveness

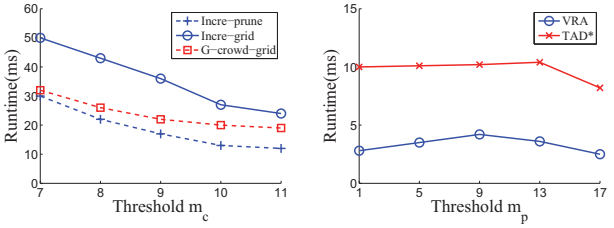
by that most participators keep evolving into next group continuously from 8:17 to 8:25. The cause of the phenomenon that the jam is separated into two groups at time 8:27 may be an efficient shunting strategy. Moreover, we can get that the two vehicle teams are scattered gradually from 8:28 to 8:35 based on the evolving groups shown in Fig. 4(b).

C. *Efficiency*

Next, we compare the performance of our framework with the discovery algorithm of gathering pattern in [16]. We denote their crowd detection and closed gathering discovery in gathering pattern [16] as **G-crowd** and **TAD*** respectively. In particular, we measure the running time of each window in different parameter settings. The results are averaged over one thousand windows. Each window slides by one minute. We omit the experiment studying k_c and k_p, since k_c and k_p only affect the number of closed crowds and participators, but have little impact on the time cost of algorithms.

Running time w.r.t. thresholds m_c and m_p. First, we evaluate the performance of two frameworks w.r.t threshold m_c and m_p. Since m_c only affects the number of crowds, we only measure the running time of closed crowds discovery using two pruning methods: a) -prune, our cluster pruning strategy; b) -grid, grid-based indexing [15][16]. We fix k_c=7 (w=8), k_p=5, d=300 meter, $|O_{DB}|$=10,000, and m_g=0.7. In Fig. 5(a), we vary m_c from 7 to 11 at fixed m_p=5. In Fig. 5(b), we vary m_p from 1 to 17 at fixed $m_c = 8$. As shown in Fig. 5(a), we can see that Incre-prune outperforms G-crowd-grid and Incre-grid methods. This may be because 1) we use two optimization principles to reduce the number of crowd candidates and prune the unnecessary validating with new clusters; 2) our cluster pruning efficiently filters the long-distance clusters as well as short-distance clusters, while grid suffers expensive cost in indexing clusters in high-speed streaming data. Since m_c affects the number of clusters that satisfy the dense group at each timestamp, the larger m_c we choose, the less number of

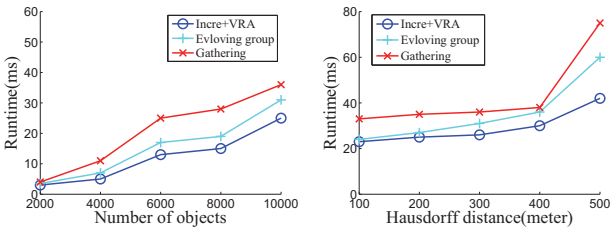
clusters satisfy this threshold. Therefore, the running time of all algorithms decreases as m_c increases. It is also easy to see that VRA exhibits much better performance than TAD* in term of CPU time in Fig. 5(b). This is because we reuse the detection process of participators in finding closed gathering as window slides, although we maintain much more closed crowd candidates. m_p would affect the number of invalid clusters in crowd. As m_p increases, more clusters are invalid, which causes process of closed gathering to terminate more quickly in each window for our framework.



(a) Threshold m_c (b) Threshold m_p

Figure 5: Running time w.r.t. threshold m_c and m_p

Running time w.r.t number of objects and hausdorff distance. Finally, we compare total running time of our framework against gathering pattern by varying the number of moving objects and hausdorff distance respectively when fix w=8, k_c= 7, m_c = 8, k_p = 5, m_p = 5, m_g=0.7. Fig. 6(a) shows the results at fixed d=300 meters. Our algorithm is superior to the discovery framework of gathering pattern (with grid indexing) in detecting closed crowds and gatherings, shown as Incre+VRA (with pruning strategy). Moreover, our overall framework also outperforms the gathering pattern in term of total running time, although our evolving group can capture more interesting patterns by considering crowds across non-consecutive time points. In particular, our framework reaches 31ms per window when $|O_{DB}|$=10,000, saving 16% CPU time compared to the gathering pattern. Fig. 6(b) shows the results w.r.t hausdorff distance at fixed $|O_{DB}|$=10,000. As expected, the running time of both algorithms increases as the distance increases. This is because the search space between new clusters and ending clusters increases although both employ pruning methods, further causing the number of crowd candidates increases at each window.



(a) Number of objects (b) Hausdorff distance

Figure 6: Running time w.r.t. number of objects and Hausdorff distance

VI. Conclusion

In this work we focus on the detection of evolving group patterns over massive-scale trajectory streams. After analyzing the requirements of stream trajectory monitoring applications, we first propose the novel concept of evolving group to capture the variety of group events and their evolving process in trajectory streams. Furthermore, we design an online discovery framework of evolving group, which contains three phases incorporating a series of optimization principles to reduce computation cost. At last we evaluate the effectiveness and efficiency of our framework compared against the state-of-the-art on a real large-scale traffic dataset.

Acknowledgment

This work is partially supported by the National Natural Science Foundation of China (nos. 61403328, 61572419 and 61502410), the Key Research & Development Project of Shandong Province (no. 2015GSF115009), and the Shandong Provincial Natural Science Foundation (no. ZR2014FQ016).

References

[1] Y. Zheng. Trajectory Data mining: an overview. ACM TIST, 2015, 6(3):1-41.
[2] S. Gaffney, P. Smyth. Trajectory clustering with mixtures of regression models. In *KDD'99*.
[3] J. G. Lee, J. Han, K. Y. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD'07*.
[4] C. S. Jensen, D. Lin, B. C. Ooi. Continuous clustering of moving objects. TKDE, 2007, 19(9):1161-1174.
[5] Z. Li, J. G. Lee, X. Li, et al. Incremental clustering for trajectories. In *DASFAA'10*.
[6] P Laube, S Imfeld. Analyzing relative motion within groups of trackable moving point objects. In *GIS'02*.
[7] P. Kalnis, N. Mamoulis. S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD'05*
[8] M. Benkert, J. Gudmundsson, F. Hbner, et al. Reporting flock patterns, Computational Geometry. 2008, 41(3):111-125.
[9] H. Jeung, H. Shen, and X. Zhou. Convoy queries in spatio-temporal databases. In *ICDE'08*.
[10] Z. Li, B. Ding, J. Han, et al. Swarm: Mining relaxed temporal moving object clusters. In *VLDB'10*.
[11] M. Vieira, P. Bakalov, V. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *GIS'09*.
[12] Y Wang, E P Lim, S Y Hwang. Efficient mining of group patterns from user movement data. DKE, 2006, 57(3):240-282.
[13] X. Li, V. Ceikute, C. S. Jensen, et al. Effective online group discovery in trajectory databases. TKDE, 2013, 25(12):2752-2766.
[14] L. A. Tang, Y. Zheng, J. Yuan, et al. On discovery of traveling companions from streaming trajectories. In *ICDE'12*.
[15] K Zheng, Y Zheng,N J Yuan, et al. On discovery of gathering patterns from trajectories. In *ICDE'13*.
[16] K Zheng, Y Zheng, N J Yuan, et al. Online discovery of gathering patterns over trajectories. TKDE, 2014, 26(8):1974-1988.
[17] H. H. Aung, K. L. Tan. Discovery of evolving convoys. In *SSDBM'10*, pp. 196-213.
[18] Martin Ester, Hans-Peter Kriegel, Jorg Sander, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD'96*. 1996, pp. 226-231.
[19] Huttenlocher D P, Klanderman G , Rucklidge W J. Comparing images using the Hausdorff distance. PAMI, 1993, 15(9):850-863.
[20] C. Gupta, S. Wang, I. Ari, M. Hao. CHAOS: A data stream analysis architecture for enterprise applications. In *CEC'09*.
[21] J. Yuan , Y. Zheng, X. Xie, et al. T-Drive: enhancing driving directions with taxi drivers' intelligence. TKDE, 2013, 25(1):220-232