Online Discovery of Evolving Groups over Massive-Scale Trajectory Streams

YANWEI YU* and RUOSHAN LAN, Ocean University of China, China
LEI CAO, Massachusetts Institute of Technology, USA
PENG SONG and YINGJIE WANG, Yantai University, China

The increasing pervasiveness of object tracking technologies leads to huge volumes of spatiotemporal data collected in the form of trajectory streams. The discovery of useful group patterns from moving objects’ movement behaviours in trajectory streams is critical for real-time applications ranging from transportation management to military surveillance. Motivated by this, we first propose a novel 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 Osaka Pedestrian data and Beijing Taxi data demonstrate its effectiveness in capturing the development, evolution, and trend of group events of moving objects in streaming context. Moreover, we propose a discovery method that efficiently supports online detection of evolving groups over massive-scale trajectory streams using a sliding window. It contains three phases along with a set of novel optimization techniques designed to minimize the computation costs. Furthermore, to scale to huge workloads over evolving streams, we extend our discovery method to a parallel framework by using a sector-based partition. Our comprehensive empirical study demonstrates that our online discovery framework is effective and efficient on real-world high-volume trajectory streams.

CCS Concepts: • Information systems → Spatial-temporal systems; Data stream mining; • Computing methodologies → Parallel algorithms.

Additional Key Words and Phrases: Moving objects, pattern mining, evolving group pattern, trajectory streams

ACM Reference Format:
Yanwei Yu, Ruoshan Lan, Lei Cao, Peng Song, and Yingjie Wang. 2022. Online Discovery of Evolving Groups over Massive-Scale Trajectory Streams. 1, 1 (August 2022), 29 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In recent years, location tracking technologies have been broadly utilized in a variety of applications ranging from traffic management to mobile social networks, which 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, such as travel-route prediction, friends recommendation, anomaly detection, and traffic control [38][21][31]. In this work, we focus on detecting a particular type of movement pattern called evolving group from massive-scale moving object trajectory streams effectively and efficiently.

*This is the corresponding author.
Evolving group pattern can be considered as a special type of group pattern that models the behavior of the moving objects that travel together over time. Techniques have been proposed in the literature to detect group patterns such as flock [3], convoy [11][12], swarm [20] and gathering [36][37]. A flock [3] consists of objects that travel together within a user-specified distance range threshold. A convoy pattern in [11][12] is defined as a set of objects that move together (always falling into the same density-based clusters) during at least k consecutive timestamps. Swarm pattern [20] is a variation of convoy pattern. It allows the moving objects to leave the swarm temporally. However, their objective is to discover groups of objects that move together in static trajectory database. Among these techniques, the gathering pattern is closest to our evolving group pattern. A gathering is defined as a sequence of density-based clusters for a period of at least k_c consecutive timestamps in which adjacent clusters are within a given distance range d. Moreover, each cluster is required to contain at least m_p dedicated members (so-called participators) who appear in at least k_p clusters, although not necessarily to be consecutive. Similar to the swarm pattern from [20], members in a gathering group also enter and leave the group. However, the gathering pattern requires at least k_c consecutive timestamps, which might result in the loss of interesting sequence of moving object clusters. Furthermore, understanding the trend and evolution of group events in streaming environments is considered more useful and helpful than simply extracting group patterns along time. For example, the causal interactions of discovered groups in time sequence can reveal the inherent relationships of groups of moving objects.

Figure 1 illustrates an example of gathering patterns. Let m_c=3 (minimal number of objects for a cluster), k_c=3 (minimal duration of a gathering), k_p=2 (minimal number of participated clusters for a participator) and m_p=3 (minimal number of participators). Suppose cluster c_3 is too far away from c_2, and c_4 is also far away from c_1, namely, distance(c_2, c_3) > d and distance(c_1, c_4) > d. From this example, two sequences of clusters {c_1, c_3, c_5} and {c_2, c_4, c_5} form two gathering candidates from t_1 to t_3. In this case, only {c_1, c_3, c_5} is a gathering pattern with participator set {o_1, o_2, o_3, o_4} because the sequence contains at least three participators at each cluster, whereas {c_2, c_4, c_5} (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 {c_6, c_7, c_9} and {c_6, c_8, c_9}. However, only {c_6, c_7, c_9} forms a gathering since each cluster includes at least three participators.

Using gathering pattern, we can get two independent gatherings from this example. This is because gathering requires the group to move in consecutive clusters during its lifetime. However, do the two gatherings have relationship with each other? We find that there are four participators \{o_1, o_2, o_3, o_4\} in the gathering \{c_1, c_3, c_5\} and five participators \{o_1, o_2, o_3, o_4, o_6\} in \{c_6, c_7, c_9\}. The four participators in the first gathering \{c_1, c_3, c_5\} still take an active part in the second gathering.

![Fig. 1. Comparison of gathering and evolving group. A small circle represents a trajectory point. A large gray dashed oval denotes a cluster.](image-url)
\( \langle c_6, c_7, c_9 \rangle \) and are also detected as participators. Additionally, these two gatherings are close to each other in both aspects of location and time. Hence gathering \( \langle c_6, c_7, c_9 \rangle \) is actually developed from \( \langle c_1, c_3, c_5 \rangle \) and grows larger. Therefore, the gathering pattern is not suitable for modeling the variational groups in which memberships evolve gradually in streaming environments.

To overcome the aforementioned shortcomings of gathering pattern, we try to define an evolving group as a dense group of objects that share common behaviors in most of the time and change gradually over time. Although gathering pattern is regarded as a dense and continuing group of individuals, we regard that it is unrealistic to use such a strict criterion of consecutive moving object clusters to track the practical group events. For example, the crowding might become “eased somewhat” at some time points in a traffic jam, which results in there is no enough dense clusters at that time. But they might again fall into a congestion soon at next crossroad. The lightweight timestamp might split the whole group of objects ponderously. The another important characteristic of evolving group is that there is no requirement for coherent membership of group over streams, namely, the members could change gradually over time. Gathering pattern also allows members to enter and leave group at any time, however, gathering does desire the stable participator to participate the group event during the whole lifetime period, while evolving group should capture the gradual change of participator in streams, i.e., the evolution of participators. Furthermore, the definition of evolving group should be suitable for capturing the group patterns from high-volume evolving streaming data. In [37], a solution is proposed to support online discovery of gatherings in incremental manner, however, it does not take the moving object evolving trajectories into account, e.g., monitoring the dynamic data of the most recent five minutes over time.

In this article, we propose a new group pattern definition of evolving group by introducing the notions of “crowd” and “aggregation” in sliding window to capture the evolving aggregation of objects over trajectory streams. Specifically, we define our ‘crowd” to model the congregation of objects over trajectory streams. Specifically, we define our ‘crowd” to model the congregation of individuals, we regard that it is unrealistic to use such a strict criterion of consecutive moving object clusters to track the practical group events. For example, the crowding might become “eased somewhat” at some time points in a traffic jam, which results in there is no enough dense clusters at that time. But they might again fall into a congestion soon at next crossroad. The lightweight timestamp might split the whole group of objects ponderously. The another important characteristic of evolving group is that there is no requirement for coherent membership of group over streams, namely, the members could change gradually over time. Gathering pattern also allows members to enter and leave group at any time, however, gathering does desire the stable participator to participate the group event during the whole lifetime period, while evolving group should capture the gradual change of participator in streams, i.e., the evolution of participators. Furthermore, the definition of evolving group should be suitable for capturing the group patterns from high-volume evolving streaming data. In [37], a solution is proposed to support online discovery of gatherings in incremental manner, however, it does not take the moving object evolving trajectories into account, e.g., monitoring the dynamic data of the most recent five minutes over time.

In this article, we propose a new group pattern definition of evolving group by introducing the notions of “crowd” and “aggregation” in sliding window to capture the evolving aggregation of objects over trajectory streams. Specifically, we define our “crowd” to model the congregation of objects in sliding window. The “aggregation” 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 “aggregation” of a window to model the core members. We further define the evolving group to refer a sequence chain of evolved “groups” that share most core members (e.g., \( m_p \)) in adjacent windows to capture the evolution of core members.

In Figure 1, let window size be 4, \( m_p = 3 \), and other parameters are same with gathering pattern. There is a “aggregation” \( \langle c_1, c_3, c_5 \rangle \) along with a “group” \( g_1 = \{o_1, o_2, o_3, o_4\} \) in window \( W_i^0 \). Suppose distance \( d(c_5, c_6) < d \), \( \langle c_3, c_5, c_6 \rangle \) also forms a “aggregation” in \( W_i^5 \) 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 \( W_i^5 \) to \( W_i^6 \). 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 “aggregation” that corresponds to the “group” \( g_3 = \{o_1, o_2, o_3, o_4, o_5\} \), which evolves from \( g_2 \) and grows larger. In window \( W_i^7 \), there is also only one “aggregation” \( \langle c_6, c_7, c_9 \rangle \) that satisfies \( m_p \) participators at each cluster, and the corresponding “group” \( g_4 = \{o_1, o_2, o_3, o_4, o_5\} \) evolves and keeps from last window. Therefore, an evolving group \( \langle g_1, g_2, g_3, g_4 \rangle \) reveals the development and trend of group events shown in the example.

The second challenge we must deal with is the design of a solution that can efficiently discover evolving groups from massive-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 real time decision-making. Hence the discovering algorithm should report the results simultaneously 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 online discovery of evolving groups. In this article, we first propose an efficient discovery framework of the evolving group, which contains three phases: 1) discover all closed “crowds” in the current window, 2) detect all
closed “aggregations” with their corresponding “groups” from the discovered crowds, and 3) update the evolving groups. Furthermore, we design a multi-threading based parallel solution to online discover the evolving groups over high-volume trajectory streams, called multi-threading based online discovery (MTOD) framework.

We conduct extensive experiments on three real world datasets (Pedestrian [35][34], Taxi [33], and citywide surveillance Traffic data in Jinan, China) to demonstrate the effectiveness and efficiency of our framework.

Contributions. The main contributions of this work are as follow: (1) We propose a novel concept of evolving group that captures the interesting group events of moving objects in the stream context. (2) Our empirical study using Pedestrian and Taxi datasets demonstrates the effectiveness of our new proposed group definition in capturing evolving groups over trajectory streams. (3) We design an efficient evolving group discovery method that efficiently discovers the evolving groups from streaming trajectories in an incremental manner. (4) We further enhance the proposed discovery method by proposing a multi-threading based parallel framework to online discover evolving groups over massive-scale moving object trajectory streams. (5) Extensive evaluation study is conducted on the real large-scale traffic datasets to evaluate the efficiency of the proposed framework in near real time.

Extension from the Conference Version. While this work is based on a conference article [15], the scope of the proposed work has been significantly extended.

- We now extend our previous algorithm to a scalable online discovery framework based on multi-threading for evolving group mining over massive-scale trajectory streams (Sec. 5).
- We now give a formal complexity analysis of our proposed algorithms (Sec. 4.4). In addition, the complexity analysis is also validated by the results of our extensive efficiency evaluation. Moreover, we also elaborate the Incr and VRA algorithms in this article.
- We significantly extend the experimental evaluation, focusing on the effectiveness of our proposed evolving group definition (Sec. 6.3). Specifically, we now evaluate the effectiveness of the proposed group definition on a new real world dataset that contains the group annotations [35][34] compared against state-of-the-art gathering pattern [37].
- We conduct an additional experimental study to demonstrate the efficiency of the newly proposed parallel framework based on multi-threading on one additional large-scale real-world dataset (the Traffic data) in Sec. 6.5.

The remainder of this article is organized as follows. We discuss related work which is related to our model in next section. We define the necessary concepts and formulate the focal problem of this paper in Section 3. Efficient methods for discovering evolving groups on archived and new arrivals of trajectory data are presented in Section 4. Online discovery solution based on multi-threading techniques is presented in Section 5. Section 6 reports the experimental observations. Section 7 concludes the article.

2 RELATED WORK

In this section, we mainly review the representative work that are related to our problem in the three areas of trajectory cluttering, group pattern mining in static trajectory databases, and in streaming trajectories.

Trajectory clustering. Gaffney et al. [7] first propose the fundamental principles of clustering trajectories based on probabilistic modeling. They consider the trajectory as a whole and represent a set of trajectories using a mixture of regression models. Then an unsupervised learning is carried out using the maximum likelihood principle. In particular, they use EM algorithm to estimate hidden parameters involved in probability models, and then determine the clusters membership. Fréchet
Online Discovery of Evolving Groups over Massive-Scale Trajectory Streams

distance is proposed to measure the similarity between curves [5][1]. In the recent years, it is widely used to measure the similarities between trajectories [26][30]. More recently, Jin et al. [13] study the problem of moving object linking based on their historical traces. They define moving object linking as a k-nearest neighbour search problem on the collection of signatures and aim at measuring the similarity of two trajectories. As pointed out by Lee et al. [18], distance measure based on whole trajectories may miss interesting common paths in sub-trajectories. Lee et al. [18] propose a partition and group framework to discover common sub-trajectory clusters in static trajectory databases. They first partition trajectories into a set of quasi-linear segments using MDL principle. Then the line segments are grouped using density-based clustering to find common sub-trajectory clusters. Lee et al. [17] succeedingly propose a hierarchical feature generation framework for trajectory classification by partitioning trajectories and exploring region-based and trajectory-based clustering. Jensen et al. [10] present a scheme that is capable of incrementally clustering moving objects by employing object dissimilarity and clustering features to improve the clustering effectiveness. Li et al. [22] further propose an incremental trajectory clustering framework that contains online micro-cluster maintenance and offline macro-cluster creation for incremental trajectory databases. Unlike the group pattern mining focused in this paper, this category of proposals only regard trajectories as sequences of line segments without considering temporal information, resulting in objects whose trajectory points fall in the same clusters may not actually move together in time domain.

Co-locating pattern discovery in static databases. One of the earliest works on co-locating pattern discovery is introduced by Laube and Imfeld [16] and further flock [3][27], convoy [11], swarm [20] and gathering [36] are studied by others. Kalnis et al. [14] 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 [28], 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. [19] 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. Recently, a kind of loose group movement pattern [29][23][24] is proposed to discover the groups with coherent members and strict density connectedness constraint among members from trajectories. Such work is to discover the clusters of coherent objects that move together in static trajectory databases, which are quit different from our evolving group mining in the context of trajectory streams.

Group pattern discovery in trajectory streams. Tang et al [25] recently propose the problem of discovering travelling companions in the context of streaming trajectories. The notion of travelling companion is as essential as convoy [11]. However, they work on incremental algorithm of pattern discovery when the trajectories of users arrive in form of data streams. Vieira et al. [27] 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. They further propose a framework that uses a lightweight grid-based structure to efficiently and incrementally process the trajectory locations to discover flock patterns in streaming spatiotemporal data. Aung and Tan [2] propose the notion of evolving convoys to better understand the states of convoys. Specifically, an evolving convoy contains both dynamic members and persisted members. As time passes, the dynamic members are allowed to move into or out of the evolving convoy, creating many stages of the same convoy. At the end, the evolving convoys with their stages are returned. Yu et al. [32] propose an online clustering over trajectory streams to discover the groups with coherent members during consecutive timestamps. They first perform density-based clustering on trajectory line segments, and
then update trajectory clusters inclemently. Zheng et al. propose an online discovery of gathering pattern [36] over trajectories in an incremental manner in [37], 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.

| Notation | Definition |
|----------|------------|
| \(o_i\)  | a moving object |
| \(O_{DB}\) | the set of all moving objects |
| \(t_i\) | timestamp at the \(i^{th}\) time point |
| \(c_{ti}\) | a cluster at timestamp \(t_i\) |
| \(C_{ti}\) | the collection of clusters at timestamp \(t_i\) |
| \(Cr\) | a crowd in a sliding 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 an aggregation |
| \(m_e\) | the support threshold for evolving group |
| \(k_g\) | the lifetime 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 \(W\) |
| \(Ag\) | an aggregation in a window |
| \(Gr\) | a group in a window |
| \(CanSet\) | the set of closed crowd candidate in a window |
| \(endclu\) | the set of ending clusters of \(CanSet\) |
| \(c_{ti}, st\) | the status flag of ending cluster \(c_{ti}\) |
| \(o_i, cnt\) | the # of clusters where \(o\) appears in a crowd |

### 3 PROBLEM DEFINITION

In this section, we first introduce the definitions of all concepts used throughout the paper, and then formally state the focal problem to be solved. The list of major symbols and notations in this article is summarized in Table 1.

Let \(O_{DB} = \{o_1, o_2, \ldots, o_n\}\) be the set of all moving objects and \(t_i\) be the timestamp at the \(i^{th}\) time points in the trajectory streams. We adopt the notion of density-based clustering [6] to define the snapshot cluster. A snapshot cluster is a group of objects with arbitrary shape and size, which are density-connected to each other at a given timestamp. \(C_{ti} = \{c_{ti}^1, c_{ti}^2, \ldots, c_{ti}^m\}\) is the collection of snapshot clusters at timestamp \(t_i\). Notice that we suppose that each snapshot is a short time interval, and the trajectory points of all moving objects in the same snapshot are received at the same timestamp, that is, the trajectory points of all moving objects generated within this interval are considered to be synchronized.
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 timestamp is called the current window, denoted as \( W_c \). We also use \( W_t^i \) to denote the window whose \( W.s = t_i \) and \( W.e = t_j \). Periodically the current window slides, causing \( W.s \) and \( W.e \) to increase by one timestamp. Now we first introduce our “crowd” concept in sliding window as following Definition 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., \( Cr = \{c_{t_a}, c_{t_b}, \ldots, c_{t_k}\} \) (\( W.s \leq t_a < t_b < \cdots < t_k \leq W.e \)), which satisfies the following three requirements:

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 \( d \times \Delta t \), where \( \Delta t \) 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 [36]. We also use the Hausdorff distance [9] to measure the distance between two clusters. Given two clusters \( c_1 \) and \( c_2 \), the Hausdorff distance \( d_H(c_1, c_2) \) between them is defined as:

\[
d_H(c_1, c_2) = \max\{\min_{q \in c_2} \min_{p \in c_1} d(p, q), \min_{q \in c_1} \min_{p \in c_2} d(p, q)\} \tag{1}
\]

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 exploring redundant crowds. Essentially the concept of crowd in sliding window can capture dense group of object clusters in most recent time. Unlike crowd in [36], we do not require that clusters in a crowd are consecutive, i.e., our “crowd” has a relaxed time restriction. Furthermore, the adjacent clusters in a crowd should satisfy the condition that their distance should be less than \( d \times \Delta t \), 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 [9], i.e., Hausdorff distance has the properties of identity, symmetry, and triangle inequality. This guarantees that 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 define the notions of participator and aggregation in sliding window environment first.

**Definition 2 (Participator).** Given a crowd \( Cr \) in the current window \( W \) and 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 (Aggregation).** Given a crowd \( Cr \) in the current window \( W \) and participator count threshold \( m_p \), \( Cr \) is called a aggregation iff each cluster of \( Cr \) includes at least \( m_p \) participators.

By Definition 2, a participator need not stay in each cluster of the crowd. As long as an object occurs in the crowd at enough timestamps, it is regarded as a participator. An aggregation also need not require a participator to occur at each timestamp in the window.

If a crowd \( Cr \) is an aggregation in the current window \( W_c \) and there is no super-crowd \( Cr' \supset Cr \) that is an aggregation, then \( Cr \) is a closed aggregation in \( W_c \). However, even if some clusters of a crowd \( Cr \) do not include enough participators, its sub-crowd may still be an aggregation if the
sub-crowd satisfies the constraint of $m_p$ participators in each cluster. Therefore, in each window we may still need to detect the closed aggregations by exploring sub-crowds space although we aim to find the closed crowds to reduce redundant crowd searching.

**Definition 4 (Group).** Given an aggregation $C_r$ in the current window $W$, the set of all participators in $C_r$ is called a group that corresponds to $C_r$ in $W$.

There is a one-to-one correspondence between a group and the aggregation in which it appears. We can also refer the concept of group to the core members of the aggregation in current window. Next we introduce the concept of evolving group that is exactly the problem which this paper studies.

**Definition 5 (Evolving Group).** Given a group $G_{r_1}$ in $W_t$, a group $G_{r_2}$ in $W_{t+1}$, a fraction support threshold $m_g$, $G_{r_2}$ is evolved from $G_{r_1}$ in streaming environments iff $|G_{r_1} \cap G_{r_2}| \geq m_g \ast \text{Min}(|G_{r_1}|, |G_{r_2}|)$, denoted as $(G_{r_1}, G_{r_2})$. Given a lifetime support threshold $k_g$, an evolving group is a chain of groups during at least $k_g$ consecutive windows, i.e., $(G_{r_{t_1}}, G_{r_{t_{a+1}}}, \ldots, G_{r_{t_b}})$, where $G_{r_{t_{i+1}}}$ is evolved from $G_{r_{t_i}}$ ($i = a, a+1, \ldots, b-1$) and $|b-a+1| \geq k_g$.

Essentially, evolving group can capture the change of core members of aggregations over time, i.e., the development of group patterns and causal relationships of groups between adjacent windows, which provides great opportunities for analyzing the impact of the different types of group events on current situation in stream context.

![Fig. 2. Architectures of DEG and MTOD frameworks.](image)

### 4 DEG: Discovering Evolving Groups

In this section, we now present our DEG method for discovering all evolving groups over trajectories in streaming window environment. Figure 2 illustrates the overall architectures of DEG and
MTOD frameworks. Basically, our DEG method includes three phases: online crowd discovery, online group detection, and evolving group updating. In the first phase, we first perform density-based clustering (DBSCAN) on the trajectory points of objects to find all the snapshot clusters at new timestamp. Thus we get the set of snapshot clusters \( C_{t_j} = \{ c^1_{t_j}, c^2_{t_j}, \ldots, c^n_{t_j} \} \) at timestamp \( t_j \). Then we try to find all closed crowds based on the clusters in the current window. The second phase aims to validate each closed crowd to see if it is or contains a closed aggregation, and then extract the corresponding group of core members. In third phase, the evolving groups are checked and updated as the window slides. MTOD framework is presented in Section 5.

4.1 Online Crowd Discovery in Sliding Window

We first introduce the incremental algorithm, called **Incre**, which discovers the closed crowds in sliding window environment. It first utilizes the obtained clusters (by DBSCAN) to construct 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 previous timestamp. Using an example-driven approach, we now describe how Incre algorithm detects the closed crowds in sliding window.

| Table 2. Example 1. snapshot clusters in sliding window |
|-----------------------------------------------|
| \( t_1 \) | \( t_2 \) | \( t_3 \) | \( t_4 \) | \( t_5 \) |
| \( c^1_{t_1} \) | \( c^1_{t_2} \) | \( c^2_{t_3} \) |
| \( c^1_{t_4} \) | \( c^4_{t_5} \) |

**EXAMPLE 1.** We use the example in Table 2 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 \( d \times \Delta t \).

We first introduce the data structures used in our algorithm to discover closed crowds in streaming windows. We use **CanSet** to store all closed crowd candidates in the current window \( W_c \). To support closure check, each candidate \( Cr \) maintains a \( Cr.endclu \) 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 the new 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 can be appended to the ending cluster \( endclu \), then \( endclu.st=match \), otherwise \( endclu.st \) is set to unmatch (abbr. unmatch). 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 \( Cr = \langle c_{t_{ia}}, \ldots, c_{t_{(b-1)}} \rangle \) (\( a \geq 0, b \geq 0 \)) in window \( W_{t_i}^{t_{j-1}} \), if \( \exists c_{t_{j+1}} \) in window \( W_{t_i}^{t_{j+1}} \) such that \( Cr_{new} = \langle c_{t_{ia}}, \ldots, c_{t_{(b-1)}, t_{j+1}} \rangle \) 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. Table 3 shows the intermediate status of crowds in Example 1 at each timestamp. Suppose \( w = 4 \) and \( k_c = 3 \). There are two clusters \( c^1_1 \) and \( c^2_1 \) at time \( t_1 \), hence...
two candidates \( \langle c_1^4 \rangle \) and \( \langle c_2^4 \rangle \) are stored in CanSet, and the ending clusters are inserted into endcu. 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 labeled to unmatch. At \( t_3 \), there is a cluster \( c_3^1 \) being close to \( c_1^1 \) and \( c_1^2 \), thus \( \langle c_1^1, c_1^2 \rangle \) and \( \langle c_1^1, c_1^3 \rangle \) are inserted into CanSet. \( c_1^3, st=match \) and \( c_1^1, st=match \) are set to match, while \( c_1^2, st \) is set to unmatch. 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 \( c_1^3 \), 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 \( W_{t_4}^l \). Hence \( c_1^3, st=match \). Moreover, we also observe that \( \langle c_1^1, c_2^1 \rangle \) 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. \( d \ast \Delta t \). \( \langle c_2^1, c_4^1 \rangle \) 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 \( c_3^1 \) must not form a closed crowd in the current window. Therefore, we set \( c_2^1, st=match \) and \( c_3^1, st=match \) directly instead of validating the candidates with \( c_4^1 \) again. We can further deduce the following corollary.

**Corollary 1.** Given a crowd candidate \( Cr = \langle c_{i_{a,b}} \rangle \) \((a \geq 0, b \geq 1)\) in window \( W_{t_i}^l \), if \( c_{i_{a,b}} \) is already included in a closed crowd candidate in \( W_{t_i}^{l_{j+1}} \), then \( Cr + \langle c_{i_{j+1}} \rangle \) must not form a closed crowd in \( W_{t_i}^{l_{j+1}} \).

Then we can quickly detect all closed crowds in \( W_{t_i}^{l_{j+1}} \). Namely, the candidates that end with clusters at \( t_3 \) or clusters whose status are unmatch, and have number of clusters not less than \( k_c(3) \). So \( \langle c_2^1, c_3^1, c_4^1 \rangle \) and \( \langle c_1^1, c_2^1, c_4^1 \rangle \) are reported as closed crowds in final.

| Timestamp | CanSet | endcu   | status | CloCr   |
|-----------|--------|---------|--------|---------|
| \( t_1 \) | \( \langle c_1^1 \rangle; \langle c_1^2 \rangle \) | \( c_1^1; c_1^2 \) | \( c_1^1, st=unmatch \) | \( c_1^1, st=unmatch \) |
| \( t_2 \) | \( \langle c_1^2 \rangle; \langle c_1^1 \rangle; \langle c_2^1 \rangle \) | \( c_1^1; c_1^2 \) | \( c_1^2, st=unmatch \) | \( c_1^1, st=match \) |
| \( t_3 \) | \( \langle c_1^1, c_1^2, c_1^3 \rangle; \langle c_1^1, c_1^4 \rangle; \langle c_1^2, c_1^3 \rangle; \langle c_1^2, c_1^4 \rangle; \langle c_1^3, c_1^4 \rangle; \langle c_1^1 \rangle; \langle c_1^3 \rangle \) | \( c_1^1; c_1^2 \) | \( c_1^1, st=match \) | (continued) |
| \( t_4 \) | (continued) | (continued) | (continued) | (continued) |

Table 3. Illustration of closed crowd discovery
After $t_5$ arrives, the window slides from $W_{t_4}^{f_4}$ to $W_{t_5}^{f_5}$. Since time $t_1$ has expired, all clusters at $t_1$ are removed from CanSet and endclus, as shown in Table 3. But we can see that some sub-crowds of existing candidates 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, $(c_1^3, c_2^3)$ is reserved, while other sub-crowd candidates are removed from CanSet. At $t_5$, there are two clusters $c_1^5$ and $c_2^5$. For validating these two clusters, we need to store two copies of CanSet, endclus and their status. For $c_1^5$, we generate a new crowd candidate $(c_1^5, c_2^3, c_4^3, c_5^3)$ since $c_5^3$ is near $c_4^3$, and the statuses of $c_1^3$, $c_2^3$ and $c_5^3$ are labeled to match. For $c_2^5$, since $c_2^5$ is far from $c_1^5$ but is close to $c_3^2$, we generate a new crowd candidate $(c_2^5, c_3^2, c_5^3)$ to insert into CanSet, and set $c_5^3$, st to unmatched, $c_3^2$, st and $c_5^3$, st to match.

Again, we can output the closed crowds from CanSet in $W_{t_5}^{f_5}$, $(c_1^3, c_2^3, c_4^3, c_5^3)$ and $(c_2^5, c_3^2, c_5^3)$, namely, candidates ending with clusters at $t_5$ or clusters whose st is unmatched at all copies.

### 4.1.1 Optimization strategies

We next present two optimization strategies to minimize computation cost. It is easy to observe that the number of CanSet has a great impact on cost of our Incremental algorithm. We first propose the Lemma 2 to reduce the unnecessary maintained candidates.

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

**Proof.** By Definition 1, the number of clusters of a crowd in a window is not less than $k_c$, thus the number of timestamps at which absent clusters in the crowd is not greater than $w - k_c$. Here, $w = j - i + 1$. So if $b > j - i + 1 - k_c$, there are no clusters in $Cr$ for at least $w - k_c$ timestamps, namely $Cr$ is not a crowd in $W_{t_i}^{f_i}$.

Moreover, $Cr$ will not be updated to a crowd in next window since the gap between $c_{t_{j-1}}$ and its next connected cluster must be greater than $w - k_c$. Therefore, $Cr$ will never be a part of a crowd in future windows even if all clusters in $Cr$ expire with window sliding.

Applying Lemma 2, the optimized CanSet in window $W_{t_5}^{f_5}$ is shown as last row in Table 3.

Furthermore, we also observe that the order of candidates in CanSet and their 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 crowd candidate $Cr = \langle c_{t_{i-1}}, \ldots, c_{t_{j-1}} \rangle$ ($a \geq 0, b \geq 0$) in window $W_{t_i}^{f_i}$, and a new cluster $c_{t_{j+1}}$, if $d_H(c_{t_{j-b}}, c_{t_{j+1}}) \leq d * (b + 1)$, then there is 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 the corresponding ending clusters in last time first order in endclus. 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 set to match if they are in endclus copy, i.e., we will skip the candidates that end with the clusters in next traversal on CanSet for $c_{new}$.

### 4.1.2 Cluster pruning strategy

Indexing clusters with R-tree or grid can improve the efficiency of discovery algorithm [37]. However the indexing 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 cluster pruning strategy to reduce the number of Hausdorff distance computation.

Specifically, 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 \( \Delta t \), if the distance between \( m_i \) and \( m_j \) is greater than \( r_i + r_j + d \times \Delta t \), then the Hausdorff distance between \( c_i \) and \( c_j \) must be greater than \( d \times \Delta t \).

**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 \( \Delta t \), if the distance between \( m_i \) and \( m_j \) is not greater than \( d \times \Delta t - \max\{r_1, r_2\} \), then the Hausdorff distance between \( c_i \) and \( c_j \) must not be greater than \( d \times \Delta t \).

![Fig. 3. Examples of pruning cluster](image)

As shown in Fig. 3(a), the smallest minimum distance from points in \( c_1 \) to \( c_2 \) must be larger than \( \text{dist} \), similarly, the smallest minimum distance from points in \( c_2 \) to \( c_1 \) also must be larger than \( \text{dist} \). Thus \( d_H(c_1, c_2) > \text{dist} \). Therefore, if \( |m_1 - m_2| > r_1 + r_2 + d \times \Delta t \), namely, \( \text{dist} > d \times \Delta t \), then \( d_H(c_1, c_2) > \text{dist} > d \times \Delta t \).

In Fig. 3(b), the maximum minimum distance from points in \( c_2 \) to \( c_1 \) must be less than \( r_2 + |m_1 - m_2| \), similarly, the maximum minimum distance from points in \( c_1 \) to \( c_2 \) must be less than \( r_1 + |m_1 - m_2| \). Hence, \( d_H(c_1, c_2) < \max\{r_1, r_2\} + |m_1 - m_2| \). If \( |m_1 - m_2| < d \times \Delta t - \max\{r_1, r_2\} \), namely, \( \max\{r_1, r_2\} + |m_1 - m_2| < d \times \Delta t \), then \( d_H(c_1, c_2) < d \times \Delta t \).

By the pruning rules, we first calculate the distance between the mean centers to determine whether we need to continue to compute the Hausdorff distance for validating the final results. Obviously, only the pruning rules that are apart between \( d \times \Delta t - \max\{r_i, r_j\} \) and \( r_i + r_j + d \times \Delta t \) need to be calculated the Hausdorff distances.

### 4.1.3 Incre Algorithm

Algorithm 1 shows the pseudo-code of our Incre algorithm. First, Incre 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). \( \text{Cr.endclu.t} \) denotes the timestamp of ending cluster of candidate \( \text{Cr} \). For a new cluster \( c_{i_j} \), we maintain a copy of CanSet, denoted as \( c_{i_j}.\text{CanSet} \). Each candidate whose ending cluster is unchecked would be validated with \( c_{i_j} \), as shown in lines 9-16. If \( c_{i_j} \) is close to \( \text{Cr.endclu} \), a new candidate is generated and inserted into CanSet, and all ending clusters in \( \text{Cr} \) are set to match (lines 11-14). \( \text{Cr.allendclu} \) stands for all ending clusters in \( \text{Cr} \). 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). \( Cr.\text{endclu.allst} = \text{unmth} \) denotes the statuses of \( Cr.\text{endclu} \) are \text{unmth} in CanSet copies of all new clusters.

**Algorithm 1** Incre algorithm for Crowd Discovery

**Input:** the current window \( W^{t_f}_{t_i}, C_{t_f}, k_c, m_c, d \)

**Output:** Closed crowds CloCr

1. \( CloCr \leftarrow \emptyset; \text{CanSet} \leftarrow W^{t_{f-1}}_{t_{i-1}} \text{CanSet}; \)
2. for each \( Cr \in \text{CanSet} \) do
3. Remove \( c_{t_{i-1}} \) from \( Cr; \) //delete clusters at time \( t_{i-1} \)
4. if \( |t_{j-1} - Cr.\text{endclu.t}| > j - i + 1 - k_c \) then
5. Remove \( Cr \) from CanSet;
6. \( \text{CanSetCopy} \leftarrow \text{CanSet}; \)
7. for each \( c_{t_j} \in C_{t_j} \) do
8. \( c_{t_j}.\text{CanSet} \leftarrow \text{CanSetCopy}; \) // a copy of CanSet
9. for each \( Cr \in C_{t_j}.\text{CanSet} \) do
10. if \( (Cr.\text{endclu.st} == \text{uncheck}) \) then
11. if \( (d_H(c_{t_j}, Cr.\text{endclu}) \leq d * \Delta t) \) then
12. Insert \( Cr + \langle c_{t_j} \rangle \) into CanSet;
13. \( Cr.\text{endclu.st} \leftarrow \text{match}; \)
14. \( Cr.\text{allendclu.st} \leftarrow \text{match}; \)
15. else
16. \( Cr.\text{endclu.st} \leftarrow \text{unmth}; \)
17. for each \( Cr \in \text{CanSet} \) do
18. if \( ((Cr.\text{endclu.t} == t_j) \& Cr.\text{endclu.allst} == \text{unmth}) \& Cr.\text{len} \geq k_c \) then
19. \( CloCr \leftarrow CloCr \cup Cr; \)

### 4.2 Online Group Extraction in Sliding Window

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

| \( t_i \) | \( t_j \) | \( t_x \) | \( t_z \) | \( t_6 \) | \( t_7 \) | \( t_8 \) |
|---|---|---|---|---|---|---|
| \( t_1 \) | \( t_2 \) | \( t_3 \) | \( t_4 \) | \( t_5 \) | \( t_6 \) | \( t_8 \) |
| \( a_1 \) | \( a_1 \) | \( a_1 \) | \( a_1 \) | \( a_1 \) | \( a_1 \) | \( a_1 \) |
| \( a_2 \) | \( a_2 \) | \( a_2 \) | \( a_2 \) | \( a_2 \) | \( a_2 \) | \( a_2 \) |
| \( a_3 \) | \( a_3 \) | \( a_3 \) | \( a_3 \) | \( a_3 \) | \( a_3 \) | \( a_3 \) |
| \( a_4 \) | \( a_4 \) | \( a_4 \) | \( a_4 \) | \( a_4 \) | \( a_4 \) | \( a_4 \) |
| \( a_5 \) | \( a_5 \) | \( a_5 \) | \( a_5 \) | \( a_5 \) | \( a_5 \) | \( a_5 \) |
| \( a_6 \) | \( a_6 \) | \( a_6 \) | \( a_6 \) | \( a_6 \) | \( a_6 \) | \( a_6 \) |

**Example 2.** Consider a closed crowd shown in Table 4, and let \( w = 8, k_c = 6, k_p = 5 \) and \( m_p = 3 \). There is a closed crowd \( Cr_1 = \langle c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_6}, c_{t_8} \rangle \) with 6 objects in window \( W^{t_8}_{t_1} \) that also evolves to a closed crowd \( Cr_2 = \langle c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_7}, c_{t_8} \rangle \) in \( W^{t_8}_{t_2} \).
We first verify if the crowd is an aggregation in window $W_{t_1}^{t_9}$. 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 in $C_{t_1}$ contains $m_p(3)$ participators, e.g., $c_{t_5}$ is considered as an invalid cluster since it only contains two participators. However, $\langle c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_6} \rangle$ still is a crowd in this window by removing the invalid cluster, meaning that it still may be an aggregation. Therefore, we need to further check the sub-crowd whether it is a closed aggregation. Again we get participator set from the sub-crowd. Obviously, the set of participators is a subset of that of $C_{t_1}$. Now we introduce the following lemma by the observation.

**Lemma 4.** Given a crowd $Cr$ and its corresponding participator set $O$ in window $W_{t_1}^{t_9}$, for any crowd $Cr' \subset Cr$, the corresponding participator set $O'$ of $Cr'$ is a subset of $O$.

Lemma 4 is intuitive. For any participator $o \in O'$ of $Cr'$, there must exist at least $k_p$ clusters in $Cr'$ that contain object $o$. Since $Cr' \subset Cr$, the $k_p$ clusters also must be included in $Cr$. Therefore, $o$ must be a participator of $Cr$, namely, $o \in O$.

Specifically, we use $o.cnt$ to denote the number of clusters in which $o$ appears in a crowd. For example, we get $\{o_1.cnt = 6, o_2.cnt = 6, o_3.cnt = 6, o_4.cnt = 2, o_5.cnt = 5, o_6.cnt = 5\}$ in $C_{t_1}$, 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 $C_{t_1} - \langle c_{t_5} \rangle$, we only need to verify if these participators appear in $c_{t_5}$. If yes, $o.cnt$ is reduced by 1, otherwise, $o.cnt$ stays the same. Therefore, we can fast find out the new participator set $\{o_1.cnt = 5, o_2.cnt = 6, o_3.cnt = 6, o_4.cnt = 5, o_6.cnt = 4\}$. Then we detect a closed aggregation $\langle c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_6} \rangle$ along with its group $\{o_1, o_2, o_3, o_5\}$, since each cluster contains at least $m_p(3)$ participators.

After $t_5$ arrives, $W_{t_1}^{t_5}$ slides to $W_{t_2}^{t_9}$. A new closed crowd $C_{t_2}$ in $W_{t_2}^{t_9}$ is generated based on $C_{t_1}$ in last step. First, we need to delete the information of $t_1$ from $C_{t_1}$. Similarly, we can directly use the above method to fast update participator set by removing invalid cluster $c_{t_1}$ from $C_{t_1}$. So the participator set is updated to $\{o_1.cnt = 5, o_2.cnt = 5, o_3.cnt = 5, o_4.cnt = 2, o_5.cnt = 5, o_6.cnt = 4\}$. Next, we consider how to update new cluster $c_{t_5}$ into the obtained result using minimal computation. We here add the new cluster into $C_{t_1} - \langle c_{t_5} \rangle$, namely, we verify $C_{t_2}$ in basis of the obtained participator set. Therefore, we only need to check whether the objects appear in new cluster $c_{t_5}$. Thus, the participator set is updated to $\{o_1.cnt = 5, o_2.cnt = 6, o_3.cnt = 6, o_4.cnt = 2, o_5.cnt = 6, o_6.cnt = 4\}$.

Similarly, we find that $c_{t_5}$ also does not satisfy the condition of $m_p$ participators, i.e., an invalid cluster, hence we next continue to verify the candidate using above removing method. Finally, $\langle c_{t_1}, c_{t_2}, c_{t_3}, c_{t_4}, c_{t_5}, c_{t_6} \rangle$ is reported as a closed aggregation with corresponding group $\{o_2, o_3, o_5\}$ in $W_{t_2}^{t_9}$.

Moreover, if $c_{t_5}$ was an invalid cluster, we can see that the detection falls back to obtained result in last window, i.e., $W_{t_1}^{t_5}$. Namely, we only need to directly verify the obtained result in $W_{t_1}^{t_5}$ by removing $t_1$.

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

As shown in Algorithm 2, $A_g$ denotes a closed aggregation, and $G_r$ is its corresponding group. $P_{ar}$ is the participator set with their cnts. If $Cr$ is an emerging closed crowd, we get its participator set of $Cr$ from scratch (denoted as $P_{participator}(Cr)$). Otherwise, we can fast find out the participator set of $Cr$ from $P_{ar}$ 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 aggregation 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 the 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 more
invalid cluster. If there is no more invalid cluster, \( Cr' \) and its current participators are reported as a closed aggregation 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 aggregation 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 \( W^{t_j}_{i} \), closed crowd \( Cr_1 \) and \( Par \) in \( W^{t_j-1}_{i-1} \), \( k_c, k_p, m_p \)

**Output:** aggregation \( Ag \), and corresponding group \( Gr \)

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

**Lemma 5.** Given a closed crowd \( Cr \) in the current window \( W^{t_j}_{i} \), If \( Ag \) is reported as an aggregation from \( Cr \) by VRA algorithm, then the aggregation \( Ag \) is closed in \( W^{t_j}_{i} \).

**Proof.** Lemma 5 can be proved easily. Suppose \( Ag \) is reported as an aggregation by VRA algorithm. According to the downward flow of our VRA algorithm, \( Ag \) 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 \( Ag \) which is an aggregation. Therefore, \( Ag \) is closed in \( W^{t_j}_{i} \) by closure property of aggregation described in Sec. 3. \( \square \)

4.3 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 whether 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 groups. 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 aggregation 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 evolved in current window, we say that the evolving group is a closed evolving group.

4.4 Complexity Analysis

Suppose that there are, on average, \( m \) new clusters per timestamp, \( n \) ending clusters in \( endclus \) and \( l \) closed crowd candidates in \( CanSet \) per window, and \( k \) trajectory points per cluster. The Incre algorithm constructs the closed crowd candidates by computing the Hausdorff distance between new clusters and ending clusters. Thus the complexity of Incre is \( O(mnk^2) \) in the worst case. Suppose that the ratio of pruning clusters, on average, \( \alpha \) in each window. The complexity becomes \( O((1 - \alpha)mnk^2) \) after applying the cluster pruning strategy. For each closed crowd, VRA algorithm detects the closed aggregation by verifying at most \( w - k_c \) clusters to update the participator set. All \( l \) crowd candidates may be closed in the worst case. Therefore, it worst-case complexity is \( O(l(w - k_c)k) \). For evolving group updating, we evaluate whether the new groups are evolved from the groups of last window. Therefore, the complexity is \( O(l^2) \) in the worst case of all \( l \) crowd candidates being aggregations. The overall complexity of our DEG method is \( O((1 - \alpha)mnk^2) + O(l(w - k_c)k) + O(l^2) \) in the worst case.

5 ONLINE DISCOVERY FRAMEWORK

To further drive down computation cost, we now present our multi-threading based online discovery (MTOD) framework. By proposing sector-based partition for object clusters, the MTOD framework achieves memory-sharing parallel discovery of evolving groups over massive-scale trajectory streams in near real time.

5.1 Sector-Based Partition

To further reduce the number of Hausdorff distance computation and simultaneously achieve load balancing, we propose an flexible sector-based partition of snapshot clusters at each timestamp for multi-threading parallel discovery framework.

As shown in Figure 4, we first find the center of data space and use concentric circles to divide the data space into multiple annuluses. The radii of concentric circles are determined according to the distribution of snapshot clusters. Then, in each annulus, we can determine the areas of sectors according to the number of clusters assigned to each thread. The partition is updated at each timestamp. For example, in Figure 4, the red solid lines are the dividing lines at the current timestamp, and the green dotted lines indicate the dividing lines at the last timestamp. The snapshot clusters are new clusters at the current timestamp.

The sector-based partition is more flexible than grid-based partition. we only need to adjust the radii of concentric circles and/or the boundaries of sectors to achieve load balancing at each timestamp. As shown in Figure 4, the new partitions can be easily obtained by only updating the sector boundaries at the last timestamp. We use \( P_t \) to denote a partition at the current timestamp and \( P'_t \) to denote a partition at the last timestamp. Notice that the radii of the concentric circles may need to be adjusted when the snapshot cluster distribution changes greatly.

To this end, we regard the the center of data space as the pole and the horizontal coordinate as the polar axis to construct a polar coordinate system. According to the stored mean centers and maximum radii of all new clusters, it is easy to obtain the radii of the concentric circles and the boundary lines of sectors for sector-based partition. More specifically, we first sort the polar coordinates of all cluster centers, and then, according to the number of clusters \( m \) and the number of
threads $T_m$, use the sorted polar radii of cluster centers to determine the radii of concentric circles, and use the sorted polar angles of cluster centers in the same ring to determine the boundary lines of sectors.

Therefore, the time complexity of re-partition mainly includes the transformation of cluster center coordinates, the polar radius sorting of cluster centers, the polar angle sorting of cluster centers in the same ring, and the division of concentric circles and the determination of sector boundary lines. That is, the total time complexity is $O(m + m \log m + \sum m_i \log m_j)$, where $m \log m$ is the time complexity of sorting polar radii of $m$ cluster centers, and $m_i$ is the number of clusters in $i$-th annulus. The space complexity is $O(\frac{m}{r_m} + r)$, where $r$ is the number of annuluses, storing the radii of annuluses, and $\frac{m}{r_m}$ is the number of partitions, storing the polar coordinate information of boundary lines of all sectors.

Additionally, we observe that the new clusters in a partition may not have to validate the ending clusters of last window in other partitions. To guarantee this point, we use the boundary zone to separate the annuluses and sectors. As shown in Figure 4, we use a ring of $2d$ width and a boundary zone of at least $2d$ width to separate two annuluses and two sectors respectively. The new clusters that cover the boundary zone are also checked with the ending clusters of last window in the adjacent partition area. In other cases, we never need to validate the closed crowd candidates with the new clusters in other partitions. Namely, we directly prune more clusters before applying the long-distance pruning. More specifically, new clusters are classified into two types based on the relative relationship between their locations and sector-based partitions of last timestamp. The first type of clusters are those that completely fall within a partition, that is, they have no overlap with any boundary zone. For example, $c_5$ lies completely in $P_3'$ of the last timestamp. Thus $c_5$ is a cluster of the first type. The second type of clusters are those remaining clusters, i.e., clusters that have overlap with any boundary zone or cross over any boundary zone. For example, since $c_7$ covers the boundary zone of $P_3'$ and $P_4'$, it belongs to the second type. Similarly, $c_6$ crosses over the boundary zone of $P_3'$ and $P_6'$ and also lies on the ring zone of $P_2'$. Thus, $c_6$ is also a cluster of the second type. Based on the above analysis, the new clusters of the first type only need to be checked with the ending clusters of last window which lie in the same partition. The new clusters of the second
type would be checked with the ending clusters of last window which lie in the partitions whose boundary zones overlapping with the new clusters.

The sector-based partition is not only applicable to memory-sharing parallel framework but also applied to shared-nothing distributed architecture, because the partition method can easily group the nearby new clusters with local closed crowd candidates.

5.2 Theoretic Analysis

In this section, we theoretically analyze and prove the correctness of our proposed parallel method.

Based on the analysis in Sec. 5.1, we classify the new clusters into two independent categories. Below we will analyze and prove their correctness separately. First, we can easily get following Lemma 6.

**Lemma 6.** Given a new cluster of the first type $c_1$ that completely falls in $P'_i$, for any cluster $c_2$ in $P'_j$ ($i \neq j$), then $c_1$ must have a Hausdorff distance larger than $d$ with $c_2$.

**Proof.** Lemma 6 is intuitive and can be proved according to long-distance Pruning in Rule 1. Since $c_1$ is a cluster of the first type, it lies completely in $P'_i$ without covering any boundary zone. $c_2$ belongs to $P'_j$ ($i \neq j$), thus the distance between $c_1$ and $c_2$ is the smallest when $P'_j$ is an adjacent partition of $P'_i$. Suppose that $P'_j$ is an adjacent partition of $P'_i$. By the analysis of the long-distance pruning, the $d_H(c_1, c_2)$ must be larger than $d$, because the width of boundary zone between $P'_i$ and $P'_j$ is $2d$. That is, even if $c_2$ covers a part of the boundary zone of $P'_j$, it does not cross the boundary line, thus there is still a Hausdorff distance of at least $d$ between $c_1$ and $c_2$. □

Lemma 6 indicates that a new cluster of the first type is only possible to match the ending clusters of last window which lie in the same partition because all the other clusters definitely have a distance larger than $d$.

Next, we have following Lemma 7 for the new clusters of the second type.

**Lemma 7.** Given a new cluster of the second type $c_1$ that covers the boundary zones among a set of partitions $P'_i = \{P'_1, P'_2, \ldots\}$, for any cluster $c_2$ in a partition $P'_a$, if $P'_a \notin P'_i$, then $c_1$ must have a Hausdorff distance larger than $d$ with $c_2$.

**Proof.** Lemma 7 can be proved based on Lemma 6. Since new cluster $c_1$ only covers the boundary zones of $P'_i$, it will not cover any boundary zone of $P'_a$ because $P'_a \notin P'_i$. Therefore, $c_1$ has a Hausdorff distance larger than $d$ with $c_2$ in $P'_a$ because of the same reason in Lemma 6. □

Lemma 7 implies that a new cluster of the second type is only possible to match the ending clusters of last window which lie in the partitions whose boundary zone is covered by itself. Namely, the new clusters of the second type are impossible to match the ending clusters of last window which lie in the partitions that have no overlap with them.

Our proposed MTOD framework is mainly based on the sector-based partition to achieve parallel evolving group discovery. For each new cluster of the first type, only the end clusters of last window in the same partition are verified. For each new cluster of the second type, only the end clusters of last window in the partitions whose boundary zone is covered by itself are verified. Therefore, the correctness of our proposed MTOD framework for the new clusters of the first and second types are both proved through Lemma 6 and Lemma 7.

5.3 MTOD Framework

Algorithm 3 shows the MTOD framework. First, we update the closed crowd candidates by removing the information of expired timestamp $t_{i-1}$ and the unnecessary maintained candidates (lines

, Vol. 1, No. 1, Article . Publication date: August 2022.
Algorithm 3 Multi-Threading based Online Discovery Framework

Input: the current window $W^{t_j}_{t_i}$, $C_{t_j}$, $k_c$, $m_c$, $d$, $k_p$, $m_p$
Output: groups CloGr in $W^{t_j}_{t_i}$

1: $CanSet \leftarrow W^{t_{j-1}}_{t_{i-1}}.CanSet$
2: for each $Cr \in CanSet$ do
3: \hspace{1em} Remove $c_{t_{i-1}}$ from $Cr$; //delete clusters at time $t_{i-1}$
4: \hspace{1em} if $(|t_{j-1} - Cr.endclu.t| > j - i + 1 - k_c)$ then
5: \hspace{2em} Remove $Cr$ from $CanSet$
6: $CanSetCopy \leftarrow CanSet$
7: $C_{t_j}^{[1..n]} \leftarrow Partition(C_{t_j})$
8: Create Thread for each partition;
9: thread_num $\leftarrow n$
10: begin Multi-Thread process:
11: for each $c_{t_j} \in C_{t_j}^{[1..n]}$ do
12: \hspace{1em} $P'_s \leftarrow LastPartition(c_{t_j})$
13: \hspace{1em} $c_{t_j}.CanSet \leftarrow CanSetCopy(P'_s)$
14: \hspace{1em} for each $Cr \in c_{t_j}.CanSet$ do
15: \hspace{2em} if $(Cr.endclu.st == uncheck)$ then
16: \hspace{3em} if $(d_H(c_{t_j}, Cr.endclu) \leq d * \Delta t)$ then
17: \hspace{4em} $Cr.endclu.st \leftarrow match$
18: \hspace{4em} $Cr.allendclu.st \leftarrow match$
19: \hspace{4em} if $(Cr.len \geq k_c - 1)$ then
20: \hspace{5em} $Ag, Gr \leftarrow VRA(Cr + \langle c_{t_j} \rangle)$
21: \hspace{5em} synchronized($CanSet$ and $CloGr$)
22: \hspace{5em} Insert $Cr + \langle c_{t_j} \rangle$ into $CanSet$
23: \hspace{5em} $CanSet.Cr.endclu.st \leftarrow match$
24: \hspace{5em} $CloGr \leftarrow CloGr \cup Gr$
25: else
26: \hspace{5em} $Cr.endclu.st \leftarrow unmatch$
27: end Multi-Thread process;
28: for each $Cr \in CanSet$ do
29: \hspace{1em} if $(Cr.endclu.st! = match \& Cr.len \geq k_c)$ then
30: \hspace{2em} $Ag, Gr \leftarrow VRA(Cr)$
31: \hspace{2em} $CloGr \leftarrow CloGr \cup Gr$

1-5). Second, MTOD uses sector-based partition method to divide the new clusters into $n$ partitions, and creates one thread for each partition. Next, in each thread process, we apply the proposed Incre and VRA to update the closed candidates in $CanSet$ and discover the closed aggregations and the corresponding groups, as shown in lines 10-27. Finally, for each unmatched closed crowd candidate after multi-thread process, MTOD uses VRA to check if it is a close aggregation (lines 28-31). Note that $C_{t_j}^{[1..n]}$ denotes the set of new clusters in partition $P'_s$ at the current timestamp. Function $LastPartition(c_{t_j})$ returns the partitions of last timestamp covered by $c_{t_j}$. Specifically, if $c_{t_j}$ is a cluster of the first type, $LastPartition(c_{t_j})$ returns the partition that $c_{t_j}$ falls into, otherwise, $LastPartition(c_{t_j})$ returns all the partitions $c_{t_j}$ covers. $CanSetCopy(P'_s)$ denotes the local closed crowd candidates in all partitions in $P'_s$. 
6 EXPERIMENT

6.1 Datasets

We use three real world datasets to evaluate the effectiveness and efficiency of our proposed algorithms compared against the state-of-the-art.

**Pedestrian data.** The dataset [35][34][4] contains the pedestrian position and group annotations in the ATC shopping center in Osaka, Japan. The tracking of pedestrians are done using automatic tracking systems, whereas the groups are labeled manually. In the dataset, there are two types of files, person tracking files and group files. Person tracking files contain the data for all persons that were tracked in the environment on a given day and period of time. Group files contain the group annotations for the given day. Only pedestrians in groups are listed, pedestrians walking alone are not included. The group files can be used to be compared with our detected results and then get *precision* and *recall*. The dataset contains 8 experiment days, and the data for 4 one-hour periods (10:00-11:00, 12:00-13:00, 15:00-16:00, and 19:00-20:00) is provided for each day. The time domain is split into 5 seconds of granularity in our experiments. More details about this dataset can be found in this website\(^1\).

**Taxi data.** The dataset is from T-Drive project [33] collected by Microsoft Research Asia. T-drive is a smart driving direction services based on GPS trajectories of a large number of taxis, which includes real-world trajectories generated by 30,000 taxis in Beijing in a period of 3 months. 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. The total number of points in this dataset is about 15 million and the total distance of the trajectories reaches 9 million kilometers. We divide a day into four time periods, morning and evening peak time (7:00-10:00 and 16:00-20:00), work time in morning and noon (10:00-13:00) and work time in afternoon (13:00 to 16:00). We interpolate the time domain into the granularity of minute on *Taxi* dataset.

**Traffic data.** The dataset is citywide surveillance traffic data collected in Jinan, China. This dataset contains 405,370,631 records of total 11,299,927 vehicles from 1,704 surveillance cameras over the period of August 1st, 2016 - August 31st, 2016. We also interpolate the time domain into the granularity of minute on *Traffic* dataset.

6.2 Experimental Setting

All algorithms in the experiment are implemented in Java on CHAOS stream engine [8]. All tests run on a computer equipped with Inter Xeon E5-2660 CPU (2.2GHz), 16G memory, and Windows Server 2012 operating system. CHAOS platform supports multiple-dimensional data and count-based/time-based sliding window streams. The arrival rate of the streaming data also can be dynamic tuned in CHAOS engine. In our experiments the arrival rate is fixed as 500k tuples per second.

Our experimental study focuses on evaluating the effectiveness and efficiency of proposed evolving group and corresponding discovering algorithms. Therefore, we compare our proposed algorithm against the state-of-art gathering pattern discovery algorithm [36][37] both on discovered patterns and utilized CPU time. Specifically, we evaluate the effectiveness of our framework on *Pedestrian* and *Taxi* datasets. We evaluate the efficiency and scalability of our online framework on large-scale *Taxi* and *Traffic* datasets. More specifically, we first do a pre-processing on the group files in *Pedestrian* dataset and get a list of real groups for each time period, which is denoted as $G_{true}$. The list of groups detected by our method or competitor for each time period is denoted by $G_{test}$. So a group is stated as “true positive” if it is both in $G_{test}$ and $G_{true}$. A group is stated as “false positive” if it is in $G_{test}$ but not in $G_{true}$. A group is stated as “false negative” if it is not in $G_{test}$ but in $G_{true}$. *Precision* and *recall* are then calculated based on the above metrics.

\(^1\)http://www.irc.atr.jp/sets/groups/
6.3 Effectiveness

6.3.1 Effectiveness on Pedestrian data. First, we evaluate the effectiveness of our proposed evolving group pattern on Pedestrian data compared against gathering pattern [36][37]. Figure 5 and Figure 6 show the comparison results of two patterns on workday and weekend, respectively. We set $MinPts=2$ and $Eps=1.5$ meters for DBSCAN, and set $w=14$, $k_c=10$, $m_c=2$, $k_p=8$, $m_p=2$ and $d=5$ meters for evolving group. Since moving groups in shopping center generally do not change for a short period, we set $m_g=1$ and $k_g=14$. Correspondingly, we set $k_c=28$, $m_c=2$, $k_p=16$, $m_p=2$, and $d=5$ meters for gathering pattern. As we can see, the recall of evolving group is much better than that of gathering pattern in all periods on both workday and weekend, while evolving group can also achieve similar precision to gathering. This indicates that our evolving group can capture more actual groups compared to gathering. This is because people in a group (e.g., a family) may be separated briefly when they are shopping in the shopping center. As shown in Figure 5(a), evolving group outperforms gathering pattern in term of precision in all periods but morning on workday. Because people in the shopping center are generally relative sparse in the morning of workday, thus dense crowds are easily identified using gathering pattern. However, evolving group significantly outperforms gathering in term of recall on workday morning (Figure 5(b)). This is because evolving group not only discovers the dense crowds but also captures the relaxed groups.
From Figure 6, we can see that our evolving group significantly outperforms gathering pattern in both precision and recall on weekend afternoon. This is because there are lots of people in the shopping center on weekend afternoon. Gathering pattern easily misses the actual relaxed groups in crowding scenes due to strict continuous time constraint, while evolving group just can capture such groups by ignoring some timestamps in the sliding window to fit the characteristics of groups in the real-world applications.

6.3.2 Effectiveness on Taxi data. We now evaluate the effectiveness of our proposed evolving group in case study of traffic condition on Taxi data. Intuitively, a serious traffic congestion is easily captured by a gathering pattern, because many vehicles aggregate in a dense cluster with slow speeds for a relatively long time. However, how a traffic jam is developed and formed? What is causal relationship between the contiguous short jams? What is the trend of a traffic congestion? Is the congestion becoming more serious? or being easing up? The gathering pattern is not applied to capture the evolving problems, while our proposed evolving group pattern exactly is to discover the evolving group events in dynamic trajectory streams.

In this experiment, we also divide the Taxi data into two categories: workday and weekend. We obtain the snapshot clusters at each timestamp by setting MinPts=5 and Eps=300 meters. Figure 7 shows the average number of patterns discovered by our proposed algorithm on a single day with the settings of $w=6, k_{c}=5, m_{c}=8, k_{p}=4, m_{p}=5, m_{d}=0.7$, and $d=300$ meters (i.e., a group of 5 or more core members travelling at least 5 timestamps in a 6-minute sliding window). We select the closed evolving groups with $k_{y} \geq 9$ (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 Figure 7(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. Figure 7(b) shows the number of discovered patterns on a weekend day. As we see, there are most traffic jams during morning peak time, and afternoon time is followed by. This is because, we learn that the weekend (February 2-3, 2008) approached the Spring Festival of China. Many companies arranged working days on that weekend. Therefore the results shows both characteristic of workday and weekend. However, compared to gathering, our evolving group also detects the traffic congestions during afternoon time, which is consistent with traffic conditions on the weekend afternoon before the Spring Festival in Beijing. Most of citizens go out to purchase the necessities, food and gifts for Festival in core areas of business street and shopping malls, or visit relatives and friends on weekend before the Spring Festival.

Next, we compare the average length of discovered patterns on a single workday and a weekend day. We select the closed evolving groups whose length $k_{y} \geq 15$ to make number of evolving groups be equal to the number of gatherings. From Figure 8, 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 groups is much larger than that of gatherings 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. During the afternoon on the weekend, our evolving group also senses the traffic congestions longer than gathering, reflecting the real non-smooth transportation condition.

![Fig. 7. Number of discovered patterns](image1)

![Fig. 8. Average length of discovered patterns](image2)

6.3.3 Case studies. We employ a visualized method with two cases to demonstrate the effectiveness of our evolving group. As shown in Fig. 9, 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. 9(a) and 9(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. 9(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. 9(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. 9(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 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

Fig. 9. Case studies on effectiveness
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. 9(b).

6.4 Efficiency

Next, we compare the performance of our DEG method with the discovery algorithm of gathering pattern in [37]. We denote their crowd detection and closed gathering discovery in gathering pattern [37] 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 ten thousand windows. Each window slides by one minute. Since $k_c$ and $m_c$ mainly affect the number of crowds, we only measure the running time of closed crowds discovery with respect to $k_c$ and $m_c$ using two pruning methods: a) -prune, our cluster pruning strategy; b) -grid, grid-based indexing used in [36][37].

![Fig. 10. Running time w.r.t. thresholds $k_c$ and $k_p$.](image)

**6.4.1 Running time w.r.t. thresholds $k_c$ and $k_p$.** We first evaluate the impacts of thresholds $k_c$ and $k_p$ on the performance of two methods when fix $w=10, m_c=8, m_p=5, d=300$ meter, $|O_{DB}|=10,000$, and $m_g=0.7$. Figure 10(a) shows the results of running time with respect to $k_c$ when $k_p=5$. Figure 10(b) shows the results of running time with respect to $k_p$ when $k_c=9$.

As shown in Figure 10(a), the CPU time of our Incr method increases as $k_c$ decreases. This is because $k_c$ in our evolving group definition would affect the number of closed crowd candidates stored in CanSet. The potential crowds whose length is not less than $k_c$ would certainly be maintained, resulting in more detection time of crowd candidates for smaller $k_c$. This is consistent with the above complexity analysis in Sec. 4.4. $k_c$ in gathering pattern only affects the number of detected gatherings, thus has no impact on the detection time. However, Incr with our proposed cluster pruning strategy can achieve similar performance to G-crowd, and outperforms G-crowd when $k_c=9$ in context of $w=10$.

From Figure 10(b), we can see that VRA is superior to TAD* in term of CPU time. This is because we reuse the detection process of participators in finding closed aggregation as window slides, although we maintain much more closed crowd candidates. In addition, $k_p$ has less impact on running time of both VRA and TAD* methods. This is because $k_p$ only affects the number of participators, but no impact on the time cost of detecting them.

**6.4.2 Running time w.r.t. thresholds $m_c$ and $m_p$.** Next, we evaluate the performance of two methods w.r.t threshold $m_c$ and $m_p$. We fix $k_c=7$ ($w=8$), $k_p=5$, $d=300$ meter, $|O_{DB}|=10,000$, and $m_g=0.7$. In Figure 11(a), we vary $m_c$ from 7 to 11 at fixed $m_p=5$. In Figure 11(b), we vary $m_p$ from
1 to 17 at fixed $m_c=8$. As shown in Figure 11(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 satisfying this threshold. Therefore, the running time of all algorithms decreases as $m_c$ increases. VRA again exhibits much better performance than TAD* in term of CPU time in Figure 11(b). The reason is same as the above explained. Moreover, $m_p$ would affect the number of invalid clusters in crowd. As $m_p$ increases, more clusters are invalid, which causes process of closed aggregation to terminate more quickly in each window for our framework.

6.4.3 Running time w.r.t number of objects and hausdorff distance. Next, we compare total running time of our DEG method against gathering pattern by varying the number of moving objects and Hausdorff distance respectively when fix $|O_{DB}|=10,000$, $|O_{DB}|=5$, $|O_{DB}|=10$, $|O_{DB}|=0.7$. Figure 12(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 aggregations, shown as Incre+VRA (with pruning strategy). Moreover, our overall framework DEG 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. Figure 12(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.

6.5 Scalability

Finally, we evaluate the scalability of our parallel MTOD framework on the large-scale Traffic data with respect to the number of threads and the number of objects (vehicles). We fix $w=8$, $k_c=7$, $m_c=10$, $k_p=5$, $m_p=10$, $d=1,500$ meters, $m_q=0.7$, and $k_q=10$. The results are shown in Figure 13. Random denotes multi-threading method using random distribution of new clusters. Figure 13(a) shows the results of all methods on all vehicles (around 180K) on the road network in Traffic data by varying the number of threads. Figure 13(b) shows the results of all methods with respect to the number of
vehicles when the number of threads is fixed to 16, i.e., \(|O_{DB}| \) varies from 60K to 180K. As shown in Figure 13(a), our MTOD framework consistently outperforms our proposed serial DEG algorithm and Random. In particular, MTOD achieves 10 speed up over our DEG algorithm when the number of threads equals to 8, while Random only improves up to 6 times faster than DEG algorithm. This is because our MTOD uses sector-based partition to distribute the new clusters into each thread, which quickly prunes the closed crowd candidates in non-adjacent sectors without computing. Therefore, our MTOD successfully achieves a super-linear speed-up ratio. From Figure 13(b), we can see that our MTOD framework shows better scalability than our serial DEG method and is superior to Random with respect to the number of objects. More specifically, MTOD framework significantly outperforms DEG by on average 10.6 times in runtime on all tested cases, whereas Random only improves on average 6.8 times over DEG. As explained above, randomly distributing only achieves parallel discovery of closed aggregations for each partition, but it still checks all closed crowd candidates in CanSet for each new clusters, while our MTOD directly prunes more crowd candidates before applying the long-distance pruning.
7 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. Moreover, we design an online discovery algorithm of evolving group, which contains three phases incorporating a series of optimization principles to reduce computation cost. Furthermore, we extend a multi-threading based parallel discovery framework to scale to huge-scale trajectory streams. At last we evaluate the effectiveness and efficiency of our method compared against the state-of-the-art on three real world large-scale datasets.

ACKNOWLEDGMENT

This work is partially supported by the National Natural Science Foundation of China (nos. 61773331, 61703360, 61403328 and 61502410).

REFERENCES

[1] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. International Journal of Computational Geometry & Applications 5, 01n02 (1995), 75–91.

[2] Htoo Htet Aung and Kian-Lee Tan. 2010. Discovery of evolving convoys. In International Conference on Scientific and Statistical Database Management. Springer, 196–213.

[3] Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. 2008. Reporting flock patterns. Computational Geometry 41, 3 (2008), 111–125.

[4] Drazen Brsic, Takayuki Kanda, Tetsushi Ikeda, and Takahiro Miyashita. 2013. Person tracking in large public spaces using 3-D range sensors. IEEE Transactions on Human-Machine Systems 43, 6 (2013), 522–534.

[5] Thomas Eiter and Heikki Mannila. 1994. Computing discrete Fréchet distance. Technical Report. Citeseer.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In Kdd. Vol. 96. 226–231.

[7] Scott Gaffney and Padhraic Smyth. 1999. Trajectory clustering with mixtures of regression models. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 63–72.

[8] Chetan Gupta, Song Wang, Ismail Ari, Ming Hao, Umeshwar Dayal, Abhay Mehta, Manish Marwah, and Ratnesh Sharma. 2009. Chaos: A data stream analysis architecture for enterprise applications. In Commerce and Enterprise Computing, 2009. CEC’09. IEEE Conference on, IEEE, 33–40.

[9] Daniel P. Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. 1993. Comparing images using the Hausdorff distance. IEEE Transactions on pattern analysis and machine intelligence 15, 9 (1993), 850–863.

[10] Christian S Jensen, Dan Lin, and Beng Chin Ooi. 2007. Continuous clustering of moving objects. IEEE Transactions on Knowledge and Data Engineering 19, 9 (2007).

[11] Hoyoung Jeung, Heng Tao Shen, and Xiaofang Zhou. 2008. Convoy queries in spatio-temporal databases. In Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, IEEE, 1457–1459.

[12] Hoyoung Jeung, Man Lung Yu, Xiaofang Zhou, Christian S Jensen, and Heng Tao Shen. 2008. Discovery of convoys in trajectory databases. Proceedings of the VLDB Endowment 1, 1 (2008), 1068–1080.

[13] Fengmei Jin, Wen Hua, Jiajie Xu, and Xiaofang Zhou. 2019. Moving Object Linking Based on Historical Trace. 2019 IEEE 35th International Conference on Data Engineering (ICDE) (2019), 1058–1069.

[14] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. 2005. On discovering moving clusters in spatio-temporal data. In International Symposium on Spatial and Temporal Databases, Springer, 364–381.

[15] RuoShan Lan, Yanwei Yu, Lei Cao, Peng Song, and Yingjie Wang. 2017. Discovering Evolving Moving Object Groups from Massive-Scale Trajectory Streams. In Mobile Data Management (MDM), 2017 18th IEEE International Conference on, IEEE, 256–265.

[16] Patrick Laube and Stephan Imfeld. 2002. Analyzing relative motion within groups of trackable moving point objects. In International Conference on Geographic Information Science. Springer, 132–144.

[17] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. 2008. TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. Proceedings of the VLDB Endowment 1, 1 (2008), 1081–1094.

[18] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 593–604.

[19] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. 2013. Effective online group discovery in trajectory databases. IEEE Transactions on Knowledge and Data Engineering 25, 12 (2013), 2752–2766.
[20] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. 2010. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 723–734.

[21] Zhenhui Li, Jiawei Han, Ming Ji, Lu-An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. 2011. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 4 (2011), 37.

[22] Zhenhui Li, Jae-Gil Lee, Xiaolei Li, and Jiawei Han. 2010. Incremental clustering for trajectories. In *International Conference on Database Systems for Advanced Applications*. Springer, 32–46.

[23] Elahe Naserian, Xinheng Wang, Xiaolong Xu, and Yuning Dong. 2016. Discovery of Loose Travelling Companion Patterns from Human Trajectories. In *High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, 2016 IEEE 18th International Conference on, IEEE, 1238–1245.

[24] Elahe Naserian, Xinheng Wang, Xiaolong Xu, and Yuning Dong. 2018. A framework of loose travelling companion discovery from human trajectories. *IEEE Transactions on Mobile Computing* 17, 11 (2018), 2497–2511.

[25] Lu-An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Chih-Chieh Hung, and Wen-Chih Peng. 2012. On discovery of traveling companions from streaming trajectories. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, IEEE, 186–197.

[26] Kevin Toohey and Matt Duckham. 2015. Trajectory similarity measures. *Sigspatial Special* 7, 1 (2015), 43–50.

[27] Marcos R Vieira, Petko Bakalov, and Vassilis J Tsotras. 2009. On-line discovery of flock patterns in spatio-temporal data. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, ACM, 286–295.

[28] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. 2006. Efficient mining of group patterns from user movement data. *Data & Knowledge Engineering* 57, 3 (2006), 240–282.

[29] Yuwei Wang, Ze Luo, Yan Xiong, Diann J Prosser, Scott H Newman, John Y Takekawa, and Baoping Yan. 2015. Discovering loose group movement patterns from animal trajectories. In *e-Science (e-Science), 2015 IEEE 11th International Conference on*, IEEE, 196–206.

[30] Dong Xie, Feifei Li, and Jeff M Phillips. 2017. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1478–1489.

[31] Yanwei Yu, Lei Cao, Elke A Rundensteiner, and Qin Wang. 2014. Detecting moving object outliers in massive-scale trajectory streams. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 422–431.

[32] Yanwei Yu, Qin Wang, Xiaodong Wang, Huan Wang, and Jie He. 2013. Online clustering for trajectory data stream of moving objects. *Computer science and information systems* 10, 3 (2013), 1293–1317.

[33] Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2013. T-drive: Enhancing driving directions with taxi drivers’ intelligence. *IEEE Transactions on Knowledge and Data Engineering* 25, 1 (2013), 220–232.

[34] Francesco Zanlungo, Dražen Brščić, and Takayuki Kanda. 2015. Spatial-size scaling of pedestrian groups under growing density conditions. *Physical Review E* 91, 6 (2015), 062810.

[35] Francesco Zanlungo, Tetsushi Ikeda, and Takayuki Kanda. 2014. Potential for the dynamics of pedestrians in a socially interacting group. *Physical Review E* 89, 1 (2014), 012811.

[36] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, IEEE, 242–253.

[37] Kai Zheng, Yu Zheng, Nicholas J Yuan, Shuo Shang, and Xiaofang Zhou. 2014. Online discovery of gathering patterns over trajectories. *IEEE Transactions on Knowledge and Data Engineering* 26, 8 (2014), 1974–1988.

[38] Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, 3 (2015), 29.