Online Discovery of Gathering Patterns over Trajectories

Kai Zheng *Member, IEEE,* Yu Zheng *Senior Member, IEEE,* Nicholas J. Yuan, Shuo Shang, and Xiaofang Zhou, *Senior Member, IEEE,*

Abstract—The increasing pervasiveness of location-acquisition technologies has enabled collection of huge amount of trajectories for almost any kind of moving objects. Discovering useful patterns from their movement behaviors can convey valuable knowledge to a variety of critical applications. In this light, we propose a novel concept, called *gathering*, which is a trajectory pattern modeling various group incidents such as celebrations, parades, protests, traffic jams and so on. A key observation is that these incidents typically involve large congregations of individuals, which form durable and stable areas with high density. In this work, we first develop a set of novel techniques to tackle the challenge of efficient discovery of gathering patterns on archived trajectory dataset. Afterwards, since trajectory databases are inherently dynamic in many real-world scenarios such as traffic monitoring, fleet management and battlefield surveillance, we further propose an online discovery solution by applying a series of optimization schemes, which can keep track of gathering patterns while new trajectory data arrive. Finally, the effectiveness of the proposed concepts and the efficiency of the approaches are validated by extensive experiments based on a real taxicab trajectory dataset.

Index Terms—Trajectory database, pattern mining, gathering pattern

1 Introduction

The increasing availability of location-acquisition technologies including telemetry attached on wildlife, GPS set on cars, WLAN networks, and mobile phones carried by people have enabled tracking almost any kind of moving objects, which results in huge volumes of spatio-temporal data in the form of trajectories. Such data provides the opportunity of discovering usable knowledge about movement behavior, which fosters ranges of novel applications and services [1]. For this reason, it has received great attention to perform data analysis on trajectories. In this paper, we move towards this direction and address one particular challenge to do with discovering the so-called *gathering* patterns from trajectories in an efficient manner.

Informally, a gathering represents a group event or incident that involves congregation of objects (e.g., vehicles, people, animals). Examples of gatherings may include celebrations, parades, large-scale business promotions, protests, traffic jams and other public gatherings. A gathering is expected to imply something unusual or significant happening. As such, the discovery of gatherings over trajectories can help in sensing, monitoring and predicating non-trivial group incidents in everyday life.

However, discovering the gatherings from trajectories is not an easy task, where challenges are two-fold. First, how to define the concept of gathering appropriately such that it intuitively captures the properties of the above mentioned events, while being rigid from algorithmic aspect in the mean time. Second, how to develop a solution that can discover gatherings from large scale trajectories efficiently, and more importantly, handle new data arrivals in an incremental manner. In the sequel, we will elaborate the two challenges and brief our contributions for addressing them respectively.

1.1 Challenge 1: Appropriate Model

To get some inspirations on how to choose the appropriate model for gatherings, we first review some related concepts in previous work. The problem of dense area detection or density query [2], [3] has been proposed with the objective of identifying *where* and *when* there are regions of high density. However the dense area cannot be adopted to model gatherings due to their limitations in two aspects. First, previous work typically identifies dense areas by overlaying a fixed grid on the geographical space, which might not correspond to the real shape of congregation in a gathering. Although this issue can be tackled to some extent by using a grid with finer granularity, the exponential increase in complexity makes this solution computationally infeasible. Second, a more intrinsic problem lies in that, the only criterion of a dense area is whether its a congregation of individuals exceeds a given threshold, regardless

- *K. Zheng is with the School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia. E-mail: kevinz@itee.uq.edu.au.*
- *X. Zhou is with the School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia, and is also with the School of Computer Science and Technology, Soochow University, China. E-mail: zxf@itee.uq.edu.au.*
- *Y. Zheng and N. Yuan are with Microsoft Research, Beijing 100080, China. E-mail: {yuzheng, nichuy}@microsoft.com.*
- *S. Shang is with the Department of Software Engineering, China University of Petroleum, Beijing, 102200, China. E-mail: sshang@cs.aau.dk.*

Manuscript received 12 June 2013; revised 23 Aug. 2013; accepted 16 Sep. 2013. Date of publication 26 Sep. 2013; date of current version 10 July 2014. Recommended for acceptance by X Lin.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier 10.1109/TKDE.2013.160

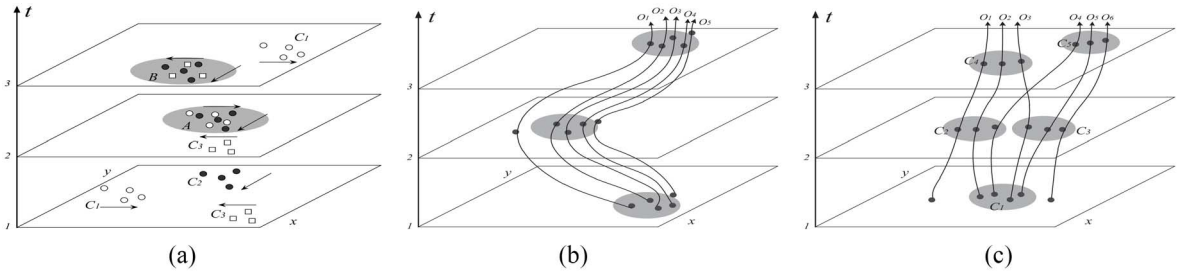

Fig. 1. Comparison of the related concepts. (a) Dense area. (b) Flock, convoy, and swarm. (c) Gathering.

of whether the individuals within the area share common behaviours. Consider Fig. 1(a) in which there are three groups of objects moving towards different directions. At $t = 2$, group c_1 and c_2 encounter and form a dense area A. After that, c_2 departures with c_1 and meets c_3 at $t = 3$, resulting in another dense area B. From this example, we can see that dense areas may be the places where individuals come by each other coincidently (e.g. major road intersections), since it does not take into account the common movements of the objects inside this area.

On the other hand, there also exist some concepts with the aim to discovering a group of objects that move together for a certain time period, such as *flock* [4]–[8], *convoy* [9], [10] and *swarm* [11]. These concepts, which we refer to as *group patterns*, can be distinguished based on how the "group" is defined and whether they require the time period to be consecutive. Specifically, a flock is a group of objects that travel together within a disc of some user-specified size for at least k consecutive timestamps. A major drawback is that a circular shape may not reflect the natural group in reality, which may result in the so-called lossy-flock problem [9]. To avoid rigid restrictions on the sizes and shapes of the group patterns, the convoy is proposed to capture generic trajectory pattern of any shape and extent by employing the density-based clustering. Instead of using a disc, a convoy requires a group of objects to be density-connected to each other during k consecutive time points. While both flock and convoy have strict requirement on consecutive time period, Li *et al*. [11] propose a more general type of trajectory pattern, called swarm, which is a cluster of objects lasting for at least k (possibly non-consecutive) timestamps. Fig. 1(b) illustrates these concepts. Let $k = 2$, the group $\langle o_2, o_3, o_4\rangle$ is a flock from t_1 to t_3. Though o_5 is an obvious company of the group, it cannot be included due to the fixed size of disc employed by the flock definition. On the other hand, a convoy can include o_5 into the group since $\langle o_2, o_3, o_4, o_5\rangle$ is density-based connected from t_1 to t_3. It is also easy to see that all the five objects form a swarm during the non-consecutive time period $\{t_1, t_3\}$.

However, using the group patterns to model gatherings is also problematic, since they all require the group to contain the same set of individuals during its lifetime. This is often unrealistic since in a practical group incidents such as celebrations or parades, members joining and leaving the event frequently is inevitable. Therefore the group patterns can be used to find the co-travellers but are not suitable for modelling those incidents in which the membership may evolve gradually. This observation motivates a more flexible concept to be developed.

Contributions. Based on the insights obtained from the above analysis, we regard a gathering as a dense and continuing group of individuals. Besides, the shape and location of the group do not change too fast, since the mobility of individuals in this group is low. Unlike the group patterns, there is no requirement for coherent membership in the gathering, i.e., members can enter and leave this group any time. However, we do desire some *dedicated members* who can commit a certain time period, though may not be consecutive, to participate the group event.

The above observations can be summarized with five key attributes, which should be possessed by the appropriate model of the gathering.

1) **Scale.** A gathering typically involves a relatively large number of individuals.
2) **Density.** Those individuals form a dense group.
3) **Durability.** It should last for a certain time period continuously.
4) **Stationariness.** The geometric properties (e.g., shape, location) of the group is relatively stable.
5) **Commitment.** At any time of the gathering, there exist several dedicated members who stick to the group for a certain time (possibly non-consecutive).

In this paper we first propose a concept called *crowd*, which captures the first four attributes. Specifically, a crowd is a sequence of density-based clusters of objects' locations which lasts for at least k_c timestamps. In order to restrict the geometrical changes of the clusters at consecutive timestamps, we adopt the widely-used Hausdorff distance [12] to measure the distance between two clusters. Then we further define the *gathering* pattern as a special kind of crowd that additionally satisfies the fifth attribute. Formally, each cluster of a gathering should contain at least m_p so-called *participators*, which refer to the objects appearing in at least k_p clusters of this gathering. These concepts can be illustrated with Fig. 1(c). Let $k_c = 3$, the two sequences of clusters $\langle c_1, c_2, c_4\rangle$ and $\langle c_1, c_3, c_4\rangle$ form two crowds. $\langle c_1, c_2, c_5\rangle$ ($\langle c_1, c_3, c_5\rangle$) is not a crowd since c_5 is too far away from $c_2(c_3)$. Let $k_p = 2$, $m_p = 3$, then only $\langle c_1, c_2, c_4\rangle$ is a gathering since it contains three participators all the time. We will re-visit this example with more explanations in Section 3.

1.2 Challenge 2: Efficient Discovery Algorithm

Now another question is: can we simply apply or extend the algorithms for group pattern mining to discover the gathering patterns? Apparently the solutions for detecting flocks cannot work since they can only find the group

within a fixed disc. The moving cluster algorithm [13] repeatedly appends a cluster of the next timestamp as long as it shares enough common objects with the current cluster. The CuTS algorithm [9] firstly clusters the simplified trajectories to obtain convoy candidates, and then applies the moving cluster algorithm to get the correct results. Both of them do not apply to our problem, since we do not require any two consecutive clusters to share common objects. Last, the *ObjectGrowth* algorithm [11] basically tries to enumerate all subsets of the object set and checks if it is a swarm. To keep the computation complexity tractable, they propose apriori pruning, backward pruning and forward closure checking to reduce the search space significantly. Nevertheless we cannot borrow these techniques either, since the gathering pattern does not have the *downward closure* property, as will be demonstrated later in Section 4.

Naturally, the solution for discovering the gatherings from trajectories can be divided into two phases: finding all the crowds over the trajectory data and validate each crowd to see if it is a gathering. Both of them can raise efficiency issues.

For the crowd detection phase, one challenge is how to find all the pairs of clusters, the Hausdorff distances between which do not exceed a given threshold, at consecutive time points efficiently. Given the quadratic complexity of Hausdorff distance computation and enormous clusters at each time instant, testing all possible pairs in the brute-force manner will render the discovery process prohibitively time consuming. In addition, we also need to take care of the redundancy problem since many crowds have containment relationship. In such cases, should we output all of them or does it suffice to just keep a subset of them?

As for the second phase, a major issue is what action should be taken whenever a crowd fails to be a gathering. Obviously, it is not wise to validate all the subsequences of the crowd due to the exponential complexity. Therefore, a smarter algorithm that examines only a few subsequences yet guarantees the correct results is desired.

Our previous work [14] has demonstrated the effectiveness and efficiency of the proposed techniques for discovering gatherings on static trajectory dataset. However, in many real applications trajectory data keep coming into the database or server for immediate analysis. With the growth of the database size, if we apply the whole process from scratch whenever new trajectory data arrive, eventually the algorithm cannot catch up with the speed of trajectory update. Therefore it is critical to develop an online algorithm to monitor the gathering patterns.

Contributions. To speed up the crowd detection process, we explore different spatial indexing techniques, namely R-tree and grid index, to organize the clusters at each time point. By this means, a large portion of cluster pairs can be safely pruned and the left candidates can also be refined at lower costs without knowing the exact Hausdorff distances. Besides, with the observation of *downward closure property* of crowds, we propose an efficient growth-style algorithm to produce the *closed crowds* only, i.e., the ones with no super-crowds.

In the gathering discovery phase, we propose a *test-and-divide* algorithm, which splits the whole crowds into subsequences by removing the *invalid clusters*, i.e., the ones with not enough participators, and tests each subsequence recursively. Since repetitively counting the occurrences of a large number of objects in a long crowd can still be lengthy, we build *bit vector signature* for each object in the crowd and apply the fast bit operations to count its occurrence. More importantly, the bit vector signatures only need to be constructed once and can be re-used by all the recursive procedures.

Lastly to meet the online monitoring requirement in dynamic environment, we propose a series of optimization mechanisms for each phase of the discovery process, including buddy-based clustering algorithm, fast online crowd detection without index and incremental gathering update algorithm, which collectively form an efficient online discovery solution over streaming trajectories.

The remainder of this paper is organized as follows. We firstly review the related work on several different research topics in Section 2. Then we define the necessary concepts and formulate the focal problem of this paper in Section 3. Efficient solutions for discovering gatherings on archived trajectory data are presented in Section 4, followed by the online monitoring algorithms on dynamic trajectory data proposed in Section 5 Section 6 reports the experimental observations, and Section 7 concludes the paper.

2 Related Work

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

Trajectory clustering techniques aim to find groups of moving object trajectories that are close to each other and have similar geometric shapes. Gaffney *et al*. [15], [16] propose trajectory clustering methods based on probabilistic modelling of a set of trajectories. As pointed out by Lee *et al*. [17], distance measure based on whole trajectories may miss interesting common paths in sub-trajectories. Motivated by this, Lee *et al*. [17] designed a partition-and-group framework, which partitions trajectories into line segments and then build groups for those close segments. More recently, Li *et al*. [18] further study the efficient algorithms for maintaining and updating the clusters when trajectories are received incrementally. Different with the group pattern mining and our work, this category of proposals does not consider the temporal aspects of the trajectories. As such, moving objects whose trajectories are in the same cluster may not actually stay together temporally.

There are also a bunch of work on mining frequent sequential pattern from moving object trajectories, which is a sequence of locations whose support is not less than a support threshold. Mamoulis *et al*. [19] are the first to investigate mining periodic movement patterns by representing trajectories as sequence of fixed regions. Giannotti *et al*. [20] study the problem of mining T-pattern, which is a sequence of temporally annotated points and propose the ROI based method to approximate a trajectory as a sequence of symbols. Liu *et al*. [21] propose to extract frequent trajectory patterns using RF tags arrays. In [22] and [23], existing sequential pattern mining algorithms are

adopted to discover frequent path segments or sequences of points. In a more recent work [24], the authors utilize the historical trajectory data to find the frequent travelling paths in-between any two consecutive locations of a given query trajectory, and then use these patterns to predicate the destination for the query trajectory. However, mining frequent travelling pattern is quite different from our problem since the moving objects that contribute the pattern may travel at different time.

Recently, a number of trajectory outlier detection algorithms have been developed, with the objective of identifying suspicious moving objects automatically. Knorr *et al*. [25] apply a distance-based algorithm, where the distance is defined between the summary information of two whole trajectories such as directions, starting (ending) points or velocities. The methodology proposed by Li *et al*. [26] is based on classification. By transforming a set of common patterns, called *motifs*, into a feature vector, the trajectories are labelled with either "normal" or "abnormal" by a classifier. Later, Lee *et al*. [27] design a *partition-and-detection* framework for detecting trajectory outliers. In their approach, they first partition the trajectories into small segments and then use both distance and density to detect abnormal sub-trajectories. Compared to [26], [27] does not require a training set, which may be more applicable in real scenarios. Besides, the problem of online abnormality monitoring over trajectory streams has also been studied in [28]. The authors utilize the local continuity characteristics of trajectories to build local clusters upon trajectory streams and monitor anomalies via efficient pruning strategies. However, our goal is totally different from these work. Trajectory outlier detection tries to find the trajectories that behave quite differently from most of the others, while our work attempt to discover groups of objects that may be involved in unusual events.

Dense area detection was initially presented in the data mining community as the identification of the set(s) of regions, from spatio-temporal data, that satisfy a minimum density threshold. The STING method [29] is a fixed-size grid-based approach to generate hierarchical statistical information from spatial data. Density query for moving objects is first proposed in [2]. Its objective is to find regions with the density higher than a given threshold at a time point or for a period of time in the near future. They find the general density-based queries difficult to answer efficiently and hence turn to simplified queries. Specifically, they partition the data space into disjoint cells, and the simplified density query reports cells, instead of arbitrary regions, which satisfy the query conditions. Later, Jensen *et al*. [3] defines a more delicate types of density query with desirable properties to address the answer loss problem in [2]. Some other solutions to detect dense areas are based on the identification of *local maxima* by using the techniques from computer vision [30], [31]. Common to all the above methods is that a fixed-size non-overlapping grid is employed to aggregate the values over the spatial dimensions, which might not correspond to the real shape of the underlying dense area. On the contrary, the gathering pattern in our work can capture the dense areas with arbitrary shapes.

TABLE 1
Table of Notations

Notation	Definition
$\mathcal{O}_{DB}$	moving object database
$\mathcal{T}_{DB}$	time domain of the database
o	the trajectory of a moving object
t	a time point in $\mathcal{T}_{DB}$
$o(t)$	the location of object o at time t
c_i	a snapshot cluster at time t_i
C_i	the set of snapshot clusters at time t_i
Cr	a crowd
$Cr.\tau$	the lifetime of a crowd
$d_H(P,Q)$	the Hausdorff distance between point sets P and Q
δ	the variation threshold in the definition of crowd
k_c	the lifetime threshold of a crowd
m_c	the support threshold of a crowd
$Par(Cr)$	the participator set of a crowd Cr
k_p	the lifetime threshold of a participator
m_p	the support threshold of a gathering
$B(o)$	the bit vector signature of an object o

3 Problem Definition

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

Let $\mathcal{O}_{DB} = \{o_1, o_2, \ldots, o_n\}$ be the set of all moving objects in the database and $\mathcal{T}_{DB} = \{t_1, t_2, \ldots, t_m\}$ be the time domain, where each t_i is a time point. The *trajectory* of a moving object o is represented by a polyline that is given as a finite sequence of timestamped locations during a closed time interval $[t_1, t_n]$, i.e., $o = \langle (p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n) \rangle$, where $p_i \in \Re^2$ is the geo-spatial position sampled at $t_i \in \mathcal{T}_{DB}$. For simplicity, we use $o.\tau$ to denote the lifespan of o and $o(t_i)$ to refer to the location of o at time instant t_i.

In our paper, we consider a practical trajectory database model, which assumes each trajectory may have different lengths and sampling rates (i.e., they are not synchronized in temporal aspect). Therefore, some trajectories may not have a sampled location for a given time instant t_i. In this case, we apply linear interpolation to create the virtual points p_i for those trajectories.

Now we adopt the notion of density-based clustering [32] to define the snapshot cluster. Given a distance threshold ϵ and a set of points S, the ϵ-neighborhood of a point p is defined as $N_\epsilon(p) = \{q \in S | D(p,q) \leq \epsilon\}$, where $D(\cdot)$ is the Euclidean distance between two points. A point p is *directly density-reachable* from a point q w.r.t. a given distance threshold ϵ and an integer m if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| \geq m$. A point p is called *density-reachable* from q if there is a chain of points $p_1, p_2, \ldots, p_n$ in S s.t. $p_1 = q, p_n = p$, and p_{i+1} is directly density-reachable from p_i. Then a point p is said to be *density-connected* to a point q if there exists a point $x \in S$ s.t. both p and q are density-reachable from x.

Definition 1 (Snapshot Cluster). *Given a trajectory set of moving objects $\mathcal{O}_{DB}$, a distance threshold ϵ, and an integer m, the* snapshot cluster *c_t at timestamp t is a non-empty subset of objects $\mathcal{O} \subseteq \mathcal{O}_{DB}$ satisfying the following conditions:*

1) *$\forall o_p, p_q \in \mathcal{O}$, $o_p(t)$ is density-connected to $o_q(t)$ w.r.t. ϵ and m.*

2) *$\mathcal{O}$ is maximal, i.e., if $o_q \in \mathcal{O}$ and $o_p(t)$ is density-reachable from $o_q(t)$ w.r.t. ϵ and m, then also $o_p \in \mathcal{O}$.*

A snapshot cluster is a group of objects with arbitrary shape and size, which are density-connected to each other at a given timestamp. Following the notion in DBSCAN [32], such snapshot clusters are spatially maximal so that no two of them with the same timestamp overlap in their objects. In the sequel we will abbreviate the snapshot cluster to cluster and omit the parameters m, ϵ when no ambiguity can be caused.

Definition 2 (Crowd). *Given a trajectory set of moving objects $\mathcal{O}_{DB}$, a support threshold m_c, a variation threshold δ, and a lifetime threshold k_c, a* crowd *Cr is a sequence of snapshot clusters at consecutive timestamps, i.e., $Cr = \langle c_{t_a}, c_{t_{a+1}}, \ldots, c_{t_b}\rangle$, which satisfies the following requirements:*

1) *The lifetime of Cr, denoted by $Cr.\tau$, is not less than k_c, i.e., $Cr.\tau = b - a + 1 \geq k_c$.*
2) *There should be at least m_c objects at any time, i.e., $\forall a \leq i \leq b, |c_{t_i}| \geq m_c$.*
3) *The distance between any consecutive pair of snapshot clusters is not greater than δ, i.e., $Dist(c_{t_i}, c_{t_{i+1}}) \leq \delta, \forall a \leq i \leq b-1$.*

Besides, a subsequence (supersequence) of a crowd Cr is called a sub-(super-)crowd of Cr, if it is also a crowd. Cr is said to be closed if it has no super-crowd.

Since a snapshot cluster is essentially a set of points, we adopt the Hausdorff distance [12] to measure how far two clusters are from each other. Hausdorff distance is a widely used metric for point sets in the community of computer vision and image processing. Given two sets of points P and Q, their Hausdorff distance $d_H(P, Q)$ is defined as

$$d_H(P, Q) = \max\{\max_{p\in P}\min_{q\in Q} d(p, q), \max_{q\in Q}\min_{p\in P} d(p, q)\}.$$

Informally, the Hausdorff distance is the longest distance one can be forced to travel by an adversary who chooses a point in one of the two sets, from where you must travel to the other set. As such, two clusters are close in the Hausdorff distance if their locations and shapes are similar with each other, which is exactly what we expect for the stationariness property as mentioned in Section 1.

Essentially, the concept of crowd captures all the properties of a gathering except the last one, i.e., it has no restriction on the membership. Before defining the gathering, we introduce the notion of participator first.

Definition 3 (Participator). *Given a crowd Cr, an object o is called a* participator *of Cr iff it appears in at least k_p snapshot clusters of Cr. Let $Cr(o)$ denote the set of snapshot clusters in Cr that contains object o, i.e., $Cr(o) = \{c_t \mid c_t \in Cr, o(t) \in c_t\}$. Then the participators of Cr are the object set $Par(Cr) = \{o \mid |Cr(o)| \geq k_p\}$.*

Note that a participator needs not to stay in the crowd for continuous k_p timestamps. As long as an object occurs in the crowd for long enough time, it is regarded as a participator. This kind of flexibility allows an individual to enter and leave a crowd multiple times, which is an usual phenomenon.

TABLE 2
Occurrences of the Objects in the Crowds

object	c_1	c_2	c_4	#	object	c_1	c_3	c_4	#
$\boldsymbol{o}_1$		–	–	2	o_1			–	1
$\boldsymbol{o}_2$	–	–	–	3	$\boldsymbol{o}_2$	–		–	2
$\boldsymbol{o}_3$	–		–	2	$\boldsymbol{o}_3$	–	–	–	3
$\boldsymbol{o}_4$	–	–		2	o_4	–			1
o_5	–			1	$\boldsymbol{o}_5$	–	–		2
o_6				0	o_6		–		1
# Par.	3	3	3		# Par.	3	2	2	

Definition 4 (Gathering). *A crowd Cr is called a* gathering *iff there exists at least m_p participators in each snapshot cluster of Cr, i.e., $\forall c_t \in Cr, |\{o \mid o(t) \in c_t, o \in Par(Cr)\}| \geq m_p$. A gathering is said to be closed if there is no super-crowd of Cr that is also a gathering.*

Example 1. Let's consider Fig. 1(c) again with $k_p = 2, m_p = 3$. To make our explanation clearer, we list the occurrence of each object in both crowds in Table 2. The participators are highlighted with bold symbols, and the bottom row shows the number of participators in each cluster. Then it is easy to see that $\langle c_1, c_2, c_4\rangle$ satisfies the support threshold at every time instant, while $\langle c_1, c_3, c_4\rangle$ only has three participators in c_1.

Problem Statement. Given a trajectory set of moving objects $\mathcal{O}_{DB}$, two support thresholds m_c, m_p, two lifetime thresholds k_c, k_p, and a variation threshold δ, our goal is to find all the closed gatherings from $\mathcal{O}_{DB}$.

4 Discovering Closed Gathering

In this section, we will present our framework for discovering all closed gatherings from a trajectory database. Basically, our framework consists of three phases: snapshot clustering, crowd detection and gathering discovery. In the first phase, we perform density-based clustering on the trajectories of objects at each time point in $\mathcal{T}_{DB}$ to find all the snapshot clusters. To reduce the cost incurred by clustering, we can apply the techniques in [9], which simplifies the original trajectories first by the Douglas-Peucker algorithm and then perform clustering on the line segments. Each cluster of line segments contains the objects that are possible to form a snapshot cluster at some time point. Finding snapshot clusters on such a set of objects is much more efficient than on the whole object set directly. The details of this phase are omitted due to space limitation, and it finally outputs a database of snapshot clusters $C_{DB} = \{C_{t_1}, C_{t_2}, \ldots, C_{t_n}\}$.

The second phase aims to find all the closed crowds from C_{DB}, while the third phase will validate each closed crowd to see if it is or contains closed gathering(s). In the next two subsections, we will elaborate our proposed techniques for improving the performance of these two phases respectively. The last subsection will discuss how to handle the new data arrivals more efficiently.

4.1 Crowd Detection

It is easy to observe that the crowd satisfies the *downward closure* property, which means any l-length subsequence of a crowd ($l \geq k_c$) is also a crowd, making it redundant to output all the sub-crowds. More importantly, gatherings

Algorithm 1: Discovering Closed Crowds

```
Input: C_DB, m_c, k_c, δ
1  V_cc ← ∅; // set of closed crowds
2  V ← ∅; // set of current crowd candidates
3  for t_i = t_1 to t_n do
4      R ← ∅;
5      for each crowd candidate Cr ∈ V do
6          c_{t_{i-1}} ← the last snapshot cluster of Cr;
7          C'_{t_i} ← RangeSearch(c_{t_{i-1}}, C_{t_i}, δ); // find the set of
           snapshot clusters that are within δ distance to Cr
8          R ← R ∪ C'_{t_i};
9          if C'_{t_i} = ∅ then // Cr cannot be extended
10             if Cr.τ ≥ k_c then
11                 V_cc ← V_cc ∪ Cr; // Cr is a closed crowd
12         else
13             for each c_{t_i} ∈ C'_{t_i} do
14                 if |c_{t_i}| ≥ m_c then
15                     Cr' ← append c_{t_i} to Cr;
16                     V ← V ∪ Cr';
17         Remove Cr from V;
18     Insert C_{t_i} \ R into V; // the snapshot clusters than cannot
       be appended to any current crowd candidate will become
       new crowd candidates
19 return V_cc;
```

detected from a non-closed crowd is not guaranteed to be closed since there may exist longer gatherings in its super-crowds. Therefore, instead of finding all the crowds, we only detect the closed crowds in this phase. At first glance, this needs to check every supersequence for a crowd in order to decide whether it is closed. However, according to the following lemma, checking the supersequence of a crowd by appending one more snapshot cluster suffices.

Lemma 1. *Given a crowd* $Cr = \{c_{t_i}, c_{t_{i+1}}, \ldots, c_{t_j}\}$, *if* $\nexists c_{t_{j+1}} \in C_{t_{j+1}}$, *s.t. appending* $c_{t_{j+1}}$ *to Cr will generate a new crowd, then Cr is a closed crowd. Otherwise, Cr is not closed.*

The proof of this lemma is omitted due to its straightforwardness. Based on this lemma, we can detect the closed crowds by incrementally appending the snapshot clusters at the next time point to the current set of crowd candidates (denoted as $\mathcal{V}$). Algorithm 1 outlines this process. At each timestamp, we check the last cluster of each crowd candidate to see if it can be extended by appending one more cluster. If so, the extended crowd candidates are inserted back to $\mathcal{V}$ as new candidates. Otherwise, we can conclude it is either a closed crowd (if the length is not smaller than k_c) based on Lemma 1, or not a crowd at all. Note that, at any timestamp the clusters (denoted by R) that cannot be appended to any existing crowd candidate should also be regarded as a new candidate, since it is possible to grow into a crowd later.

It is easy to see that the most costly part in Algorithm 1 is the procedure *RangeSearch*(), which looks for the clusters from the cluster set at current timestamp C_{t_c}, whose Hausdorff distance with c_i is not greater than δ. A naive implementation of this procedure is to calculate $d_H(c_i, c_j)$ for each $c_j \in C_{t_c}$. Apparently, a single calculation of $d_H(c_i, c_j)$ requires $O(|c_i||c_j|)$ time, and it should be performed over all pairs between current crowd candidates and the clusters at the current time point. This will make the overall computation prohibitively expensive for a large dataset. To address this issue, we will explore spatial indexing techniques to organize the clusters and speed up the search process.

4.1.1 Indexing Clusters with R-tree

Actually we do not need the exact Hausdorff distance between two clusters. Instead, it suffices to just know whether their distance is below or above δ. Let $M(c)$ denote the minimum bounding rectangle (MBR) of cluster c and $d_{min}(,)$ the minimum distance between two rectangles. The following lemma holds naturally.

Lemma 2. *Given two clusters* c_i *and* c_j, $d_{min}(M(c_i), M(c_j)) \leq d_H(c_i, c_j)$

The proof of this lemma is omitted due to its straightforwardness. Based on this lemma, we firstly retrieve a candidate set of clusters from C_{t_c} whose minimum distance with c_i is not greater than δ and then refine the candidates to get the final results. To support efficient candidate search, we index the MBRs of the clusters in C by a R-tree, and then perform a window query against the R-tree, in which the window is the enlarged MBR of c_i by δ. Obviously, clusters contained in the nodes not overlapping with the window are not candidates.

However, d_{min} is rather a loose lower bound for the Hausdorff distance since the latter is the maximum of minimum distance from one cluster to the other. The following lemma provides a tighter lower bound for the Hausdorff distance.

Lemma 3. *Let* $M.l_a$ *denote the a-th side of a rectangle* M ($a = 1, 2, 3, 4$). *Define the distance function* d_{side} *to be*

$$d_{side}(M(c_i), M(c_j)) = \max_{a \in [1,4]} d_{min}(M(c_i).l_a, M(c_j))^1.$$

Then we have $d_{side}(M(c_i), M(c_j)) \leq d_H(c_i, c_j)$.

Proof. Let p_a be the point of the cluster c_i that lies on the side $M(c_i).l_a$. Naturally, $d_{min}(M(c_i).l_a, M(c_j)) \leq d_{min}(p_a, M(c_j))$. From the definition of Hausdorff distance, $d_{min}(p_a, M(c_j)) \leq d_H(c_i, c_j)$ since $p_a \in c_i$. As such, $d_{min}(M(c_i).l_a, M(c_j)) \leq d_H(c_i, c_j)$, $\forall a \in [1, 4]$. By taking their maximum, d_{side} still lower bounds d_H. □

To retrieve candidates in the R-tree by utilizing d_{side}, we need slight modifications to the aforementioned window query process as follows. First we enlarge each side of $M(c_i)$ by δ to obtain four rectangles, denoted by $r_a, a = 1, 2, 3, 4$. During the traversal of R-tree, a node needs to be further examined only if it intersects with all the four rectangles.

4.1.2 Indexing Clusters with Grid

Indexing clusters with R-tree, though improving the detection performance by ruling out many disqualifying clusters, still suffers from three major drawbacks. First, an R-tree needs to be constructed and maintained for each time point, which may incur high cost. Second, since the density-based clusters may have arbitrary shapes, rectangular bounding box cannot always capture the distribution of points in a cluster, which will affect its pruning effect. Third, the brute-force refinement

1. d_{min} here is used to compute the minimum distance between a side and a rectangle since the side can be regarded as a degenerated rectangle.

is still needed to evaluate the Hausdorff distances for those candidate clusters. To address them, we propose a grid-based index for clusters. As we shall see shortly, the grid index is easier to construct since the clusters at all timestamps can share the same grid structure. More effective pruning can be performed as the composition of grid cells can approximate the shape of a cluster better. Besides, a smarter refinement algorithm can be devised by utilizing the grid index, which is able to validate a candidate without calculating the exact Hausdorff distance.

To start, we partition the whole space by a grid, each cell of which is a square with the side length equal to $\frac{\sqrt{2}}{2}\delta$. Then for each time point t, we can build a grid index G_t with two kinds of data structures by scanning the set of clusters once, namely a cell list for each cluster $c.cl$ that keeps the cells occupied by the cluster, and an inverted list for each cell $g.inv$ that stores the clusters covering this cell. Before describing the algorithm, we define the *affect region* for a cell.

Definition 5 (Affect region). *Given a cell g_{ab} locating at the a-th row and b-column of a grid G, its* affect region *is the set of cells whose minimum distance with g_{ab} is not greater than δ. More precisely, $AR(g_{ab}) = \{g_{ij} \in G \mid |i-a| \leq 2, |j-b| \leq 2,$ and $|i-a| + |j-b| < 4\}$.*

Intuitively, the affect region of a cell g may contain some point whose distance with a point in g is not greater than δ. Now given the query cluster c_i, (i.e., the last cluster of some crowd candidate) and grid index $G_{t_{i+1}}$ at the next timestamp, the procedure *RangeSearch*() of Algorithm 1 works in the pruning-refinement style, stated as follows.

In the pruning phase, we select each cell g from $c_i.cl$ and find the clusters in $C_{t_{i+1}}$ whose cell list intersects with $AR(g)$. Easy to know that, only the clusters that overlap with the affect region of every cell in $c_i.cl$ can be the candidates, since otherwise there exists at least one point in the cluster that is farther away from c_i than δ.

In the refinement phase, we will validate each candidate to determine the final results. For a candidate c_j, we first perform a set join on $c_i.cl$ and $c_j.cl$ to get their common cells. The rational behind is that the distance between any two points within the same cell cannot be greater than δ. In other words, the Hausdorff distance between the subsets of c_i and c_j that fall inside their common cells will not exceed δ. In an extreme case, if $c_i.cl = c_j.cl$, we can immediately conclude $d_H(c_i, c_j) \leq \delta$. For this reason, we just need to check the cells in their difference set, i.e., $dif(c_i.cl, c_j.cl) = (c_i.cl \cup c_j.cl) \setminus (c_i.cl \cap c_j.cl)$. For each point p within $dif(c_i.cl, c_j.cl)$, assuming $p \in c_i$ without loss of generality, we calculate its minimum distance with c_j. Note that we only need to calculate the distances between p and the points falling inside the affect region, since all the other points will definitely have distances with p greater than δ.

4.2 Gathering Discovery

In this subsection, we will discuss the algorithm to discover closed gatherings on each closed crowd obtained from the last step. It seems that we can apply the similar methodology with the crowd detection – incrementally extending a shorter cluster sequence into a longer one until it fails to be a gathering. However, the downward closure property does not hold for gatherings. In other words, a non-gathering cannot imply its super-sequences also not being gatherings. To see this, consider a crowd with four clusters $c_1 = \{o_1, o_2, o_3\}$, $c_2 = \{o_1, o_2, o_4\}$, $c_3 = \{o_1, o_3, o_4\}$, $c_4 = \{o_2, o_3, o_4\}$, and let $k_p = 3, m_p = 2$. Obviously, neither the crowd $\langle c_1, c_2, c_3\rangle$ nor $\langle c_2, c_3, c_4\rangle$ is a gathering as the number of participators in $c_2(c_3)$ is less than m (only 1). When we see their super-crowd $\langle c_1, c_2, c_3, c_4\rangle$, it is a gathering indeed. As such, for a gathering found so far, we have to check all the super-crowds in order to know if it is closed. Undoubtedly, this will incur high computation cost especially when the given crowd is a long sequence.

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
	o_1	o_1		o_1	o_1		
o_2	o_2	o_2	o_2			o_2	o_2
o_3	o_3		o_3		o_3	o_3	o_3
o_4		o_4	o_4	o_4	o_4	o_4	o_4
	o_5	o_5	o_5				
				o_6	o_6		

Fig. 2. Illustration of test-and-divide algorithm

4.2.1 Test-and-Divide Algorithm

In the sequel, we propose a test-and-divide (TAD) algorithm that can detect all the closed gatherings in a given crowd efficiently. As shown in Algorithm 2, it starts from the whole closed crowd and tests if it is a gathering. If so, as we shall prove shortly, it is a closed gathering and can be returned immediately. Otherwise, we identifies the *invalid clusters*, which does not have enough participators, and divide the crowd into several subsequences by removing these clusters (some subsequences may not be crowds as their lengths are less than k). For each subsequence that is still a crowd, we repeat the above steps again since some objects may become non-participators now due to the deletion of invalid clusters. This procedure is performed recursively until no crowd can be found any more .

Example 2. Consider a closed crowd illustrated in Fig. 2, and let $k_c = k_p = 3$, $m_c = m_p = 3$. According to the TAD algorithm, we first apply the *Test*() procedure on the whole crowd. It is easy to see that, the objects o_1, o_2, o_3, o_4, o_5 are participators w.r.t. the whole crowd. So c_5 is an invalid cluster as it only contains two participators (< 3). By removing c_5, we divide the original crowd into two sub-crowds $Cr_a = \langle c_1, c_2, c_3, c_4\rangle$ and $Cr_b = \langle c_6, c_7, c_8\rangle$. Again, we perform *Test*() recursively on Cr_a and Cr_b respectively. For Cr_a, though o_1 changes to a non-participator, all the clusters still have enough number of participators, so we output Cr_a as a gathering. For Cr_b, both o_1 and o_2 become non-participators, making all the three clusters invalid. Since we cannot get any more sub-crowds from Cr_b, the TAD algorithm will terminate.

Theorem 1. *The gatherings output by TAD are closed.*

Proof. We can prove it by contradiction. Suppose at some stage of TAD, we get a sub-crowd $Cr = \langle c_i, c_{i+1}, \ldots, c_j\rangle$ that turns out to be a gathering. According to the work flow of TAD, the reason we get Cr is that both c_{i-1} and c_{j+1} are invalid clusters. On the other hand, if there exists any super-crowd of Cr such that it is also a gathering,

Algorithm 2: Test and Divide (TAD)

```
Input: Cr, k_c, k_p, m_p
R ← ∅; // the set of closed gatherings
if Test(Cr,k_p,m_p) is true then // test if Cr is a gathering
    return Cr;
else
    C ← find the invalid clusters;
    S_cr ← Divide(Cr, C); // divide Cr by removing clusters in C
    for each Cr' ∈ S_cr do
        if Cr'.τ ≥ k_c then // if Cr' is still a crowd
            R ← R∪ TAD(Cr', k_c, k_p, m_p);
return R;
```

then at least one of c_{i-1} and c_{j+1} should have enough participators, which is contradictory to the previous claim. Therefore Cr is a closed gathering. □

4.2.2 Efficient Implementation with Bit Vector Signature

A straightforward implementation of TAD algorithm is to count the occurrence for each object in the crowd to see if it is a participator, and then check the number of participators for each cluster in the crowd. Obviously this requires $O(m \cdot Cr.\tau)$ time where m is the number of objects in Cr. Even worse, we have to perform the above operations repeatedly from scratch for each sub-crowd obtained.

For a more efficient implementation of TAD, we propose to construct a *bit vector signature* (BVS) for each object of Cr, and all the subsequent steps can be performed with fast bitwise operations. Specifically, given a crowd $Cr = \langle c_1, c_2, \ldots, c_n \rangle$, the BVS for an object $o \in Cr$ is an n-length bit vector with each bit representing the existence of o in the corresponding cluster. The BVSs of all the objects in Cr can be constructed by a single scan of the crowd. More importantly, the BVSs only need to be built once and can be used for all the recursions of TAD. Next we elaborate how to implement the two procedures *Test*() and *Divide*() in Algorithm 2 by utilizing the BVS.

Test step. With the BVS of some object o, denoted by $B(o)$, the procedure *Test*() essentially turns out to be counting the 1 bits in $B(o)$, which is also known as the Hamming weight [33] of a bit vector. While a naive method is to iterate over all the bits of $B(o)$, more efficient implementations have been well studied. One of the best solution known is based on adding the counts in a binary tree pattern [33], in which we first get the number of 1s in every 2-bit piece of $B(o)$, and then in every 4-bit piece,..., and so on so forth. The example below shows how we can get the Hamming weight of $B(o_1)$ in just 3 steps. Let $x = B(o_1)$,

1) Let $m1 = 01010101$,
   $x = (x\&m1) + ((x \gg 1)\&m1) = 01011000$
2) Let $m2 = 00110011$,
   $x = (x\&m2) + ((x \gg 2)\&m2) = 00100010$
3) Let $m4 = 00001111$,
   $x = (x\&m4) + ((x \gg 4)\&m4) = 00000100$

Now the decimal number of x is 4, which is exactly the number of 1s in $B(o_1)$. In the above operations, $m1, m2, m4$ are called *masks* and can be defined properly once the length of the bit vector is known. In general, for any bit vector with n bits, its Hamming weight can always be obtained in $\lceil \log_2(n) \rceil$ steps.

Divide step. In this step we will divide the crowd into a set of subsequences if it fails to be a gathering. Essentially this is to split the BVS of each object into a set of subvectors. It is worth pointing out, there is no need to process the BVSs of non-participators since a non-participator of a crowd must remain a non-participator in any of its sub-crowds. We also do not have to split the BVS physically, instead of which we can just use a mask to extract the desired part from the original BVS. The mask is also a bit vector having the same length as the BVS. It sets to 1 in the bits corresponding the sub-crowd, and 0 in all the other bits. By performing the AND operation on the original BVS and the mask, we get a new BVS where the bits of the desired sub-crowd are kept while all other bits are zero. For example, in Fig. 2 the mask to extract the crowds Cr_a and Cr_b are 11110000 and 00000111 respectively. As such, the *Divide*() just needs to return a set of masks, which is more compact compared to the subsequences of a crowd, and pass it to the subsequent *Test*() procedure. By this means, the *Test*() procedure can use each mask to get the BVSs of the objects in the corresponding sub-crowd directly, thus avoiding the re-construction of BVSs for each sub-crowd.

5 Monitoring Gathering Patterns

In the previous section, we have introduced the efficient algorithms for discovering the closed gathering patterns from a static trajectory archive, which has been collected and stored in a database beforehand. In many applications, such as traffic management and battlefield surveillance, trajectories of moving objects (e.g., vehicles, military units) are continuously received by sensors and immediately sent back to central servers for further analysis. It is often critical and beneficial for these applications to monitor the changes of patterns in real-time so that favourable decisions can be made as early as possible. However, the goal of efficient monitoring of gathering patterns cannot be achieved by using the techniques we have discussed before due to several reasons: 1) The simplify-and-cluster method [9] adopted for clustering is based on the assumption that all the trajectory data are available. In the online monitoring scenario, future positions of moving objects are not known a priori and thus the trajectory simplification technique is not applicable. 2) The crowd detection algorithm heavily depends on the index structure built for clusters in each snapshot to find the nearby clusters for each crowd candidate. When trajectory data arrive in streaming fashion, however, it is not realistic to construct and maintain an index structure for each new snapshot of locations. 3) The gathering discovery algorithm adopts a divide-and-conquer paradigm. It means when existing crowds get extended by new trajectory data, the gathering patterns cannot be incrementally updated and will be re-discovered from scratch.

Based on these observations, in this work we extend the existing gathering discovery framework by designing an on-line component for efficient monitoring of gathering patterns. The on-line component addresses the three major issues identified above by re-designing the three phases, namely snapshot clustering, crowd detection and gathering

discovery, to meet the monitoring purpose. We will detail each phase in the following subsections.

5.1 Travelling Buddy Based Clustering

The time cost of density-based clustering without spatial index is quadratic to the number of objects, and maintaining a spatial index like R-tree in each snapshot is also very expensive. Therefore, we have to speed up the clustering phase first in order for the on-line component to keep up with the update rate of trajectories. Despite the fact that moving objects keep on moving and updating their positions, the changes of spatial relationship among objects are gradual evolution rather than fierce mutation. In most real world applications, there exist some kinds of *co-travellers* who tend to stay close with each other for a while. For examples, couples would like to stay together on trips, military units operate in teams, families of birds, deer and other animals often move together in species migration. These moving objects form a smaller but more flexible structure, in which the object relationships are possible to be retained in a few snapshots. It is attractive to exploit this property to speed up the clustering phase since many distance calculation between individual pairs may be avoided. To this end, we utilize the concept of *travelling buddy* proposed in [34], which is defined as follows:

Definition 6 (Travelling Buddy). *Given a radius threshold r, a travelling buddy b at time t is defined as a set of moving objects satisfying: (1) $b \subseteq \mathcal{O}_{DB}$; (2) $\forall o_i \in b$, $dist(o_i(t), cen(b)) \leq r$, where $cen(b)$ is the geometry center of b. The buddy's radius γ is defined as the distance from $cen(b)$ to it's farthest member.*

The travelling buddies can be initialized by merging the objects with their nearest neighbours until the buddy's radius is larger than γ. The initialization step takes $O(n^2)$ time for n objects, but it only needs to be carried out once and the travelling buddies can be incrementally maintained upon the arrival of trajectory data. There are two kinds of operations to maintain buddies, namely split and merge.

- *Split.* When the data of a new snapshot at t_{n+1} arrive, the maintenance algorithm first updates the center of each buddy b by calculating the shift $(\Delta x_i, \Delta y_i)$ of each member between t_n and t_{n+1}: $cen_{n+1}(b) = cen_n(b) + \sum_{o_i \in b}(\Delta x_i, \Delta y_i)$. Afterwards every object $o_i \in b$ checks its distance to the buddy center and will be split out as a new buddy if the distance is greater than δ.
- *Merge.* This operation is to merge the buddies that are close to each other. If two buddies b_i and b_j are close enough to satisfy the equation:$dist(cen(b_i), cen(b_j)) + \gamma_i + \gamma_j \leq 2r$, they should be merged as a new buddy. The center of new buddy can be easily computed using the old buddy's center and size.

The travelling buddies can help reduce the number of individual objects that need to be accessed during the clustering phase, based on the following lemmas.

Lemma 4. *[34] Given a distance threshold ϵ and a density threshold μ for the density-based clustering, if a buddy's size if larger than $\mu + 1$ and the buddy radius is less than $\epsilon/2$, then all the objects in b are directly density reachable from each other. We call such kind of buddy a density-connected buddy.*

Lemma 5. *[34] Let b_i and b_j be two travelling buddies with radius γ_i and γ_j, and ϵ be the distance threshold. If $dist(cen(b_i), cen(b_j)) - \gamma_i - \gamma_j > \epsilon$, then the objects in b_i and b_j are not directly reachable.*

Lemma 6. *[34] Let b_i and b_j be two density-connected buddies and ϵ be the distance threshold. If $\exists o_i \in b_i, o_j \in b_j$ such that $dist(o_i, o_j) \leq \epsilon$, then all the objects in b_i, b_j are density connected.*

Lemma 5 implies that when searching for directly density reachable objects, if another buddy is too far away, we can safely prune all its members without further computations. On the other hand, based on Lemma 6 once we find a pair of objects in two buddies are close to each other, the two corresponding buddies must be density-connected.

The buddy-based clustering algorithm firstly updates the buddy set in a new snapshot using the split and merge operations. Then it randomly picks a buddy and checks the density connectivity with other buddies. In most cases, Lemma 5 and Lemma 6 can quickly filter out majority far-away buddies and include the close enough buddies directly during this process. Finally, the algorithm outputs the clusters when all the buddies have been processed.

5.2 On-line Closed Crowd Detection

The crowd detection algorithm proposed in the previous section relies on some spatial index to find the clusters with small Hausdorff distance efficiently. As we mentioned, construction of spatial index is a costly procedure itself and thus not practical for on-line scenarios where each new snapshot must be processed in real-time. In this part, we propose an on-line crowd detection algorithm, which can identify the closed crowds fast without any facility of spatial index. More specifically, it utilizes the technique of on-line spatial join to prune most far-away clusters and exploits the travelling buddies inside the clusters to speed up the distance evaluation in refinement.

Pruning. Once the set of clusters C_{n+1} in the new snapshot is obtained, we firstly need to identify the candidates that are likely to satisfy the distance condition with some cluster at snapshot t_n and potentially form a crowd. To do that, we represent each cluster in C_n, C_{n+1} by its minimum bounding rectangle (MBR) and expand the MBRs of C_n by δ. Then if the MBR of some cluster c_{n+1}^i overlaps with the expanded MBR of some cluster c_n^j, c_{n+1}^i is regarded as a candidate of c_n^j. It is easy to prove that, the Hausdorff distance between these two clusters will not be less than δ if their MBRs do not overlap. Now our problem becomes finding all pairs of MBRs in the two cluster sets that overlap with each other, which is exactly a spatial join operation. Despite this problem has been studied for decades, generally there are only approaches that can be used to process spatial join on-line without the availability of index: the nested loop join [35] and the plane-sweep join [36]. We adopt the latter approach for this task due its better performance in practice. The algorithm starts with sorting the MBRs in both cluster sets based on one dimension and then a sweep line

moves along this dimension from one end to the other, during which course the MBRs passed through by the sweep line are checked for overlap relationship. Since the MBRs are unsorted in the other dimension, pairs of MBRs that are far away from each other in that dimension still need to be examined. As we will show in the experiments shortly, it will not cause much performance overhead as the number of MBRs overlapping with a sweep line is usually small.

Refinement. After the pruning step, each cluster c_n^i in C_n will be associated with a list of candidate clusters in C_{n+1} that may satisfy the distance condition. The refinement step is to evaluate the actual Hausdorff distance between c_n^i and each of its candidates. Without the help of spatial index, a simple approach will have to use a nested loop to go through all the point pairs of two clusters in order to derive the exact Hausdorff distance. Recall that we utilize travelling buddies to speed up the clustering phase, so in each cluster all the buddy information are still available, which gives us opportunity to optimize the refinement process. The overall idea is to, instead of directly calculating the distance between individuals, estimate the distance range between buddies using their centers and radius and reduce unnecessary examination of individual points based on this estimation. Before presenting the detailed algorithm, we will introduce the following lemma that states the relationship between the Hausdorff distance and distance of buddies.

Lemma 7. *Let $d_{min}(b_1, b_2)$ and $d_{max}(b_1, b_2)$ be the minimum and maximum distance between two buddies, which can be calculated based on the distance between their centers minus/plus their radius. Then given a buddy b and another point set P consisting of n buddies $\{b'_1, b'_2, \ldots, b'_n\}$, the minimum distance between any pair of points from b and P is in the range of $[\min_{b' \in P} d_{min}(b, b'), \min_{b' \in P} d_{max}(b, b')]$. We denote these distance bounds as $d_{min}(b, P)$ and $d_{max}(b, P)$ respectively.*

The buddy-based Hausdorff distance evaluation, as shown in Algorithm 3, starts with calculating d_{min} and d_{max} between all pairs of buddies in C_n and C_{n+1}, and then sorts all the buddies b in C_n descendingly based on $d_{max}(b, C_{n+1})$ (Line 2). For each buddy b in this sorted list, we will evaluate its actual minimum distance with C_{n+1} and keep the current greatest value in a global variable d_h (Line 3-15). This process can stop when the list is exhausted or it is found that $d_{max}(b, C_{n+1})$ of the next-to-be visited buddy is less than d_h (Line 4-5), which means all the unvisited buddies will have no effect on final Hausdorff distance. To further speed up the process of deriving the minimum distance between each buddy b and C_{n+1}, we examine and access the individual points of each buddy b' of C_{n+1} in the order of $d_{min}(b, b')$ (Line 7-13), and maintain the current minimum distance m_b at the same time. In this way, all the buddies in C_{n+1} with $d_{min}(b, b')$ greater than the current best result can be pruned safely (Line 9-10).

Algorithm 3 still needs to go over all pairs of buddies in the two clusters and may access the individual points of all buddies in the worst case. In practice this algorithm can achieve better efficiency than the nested loop method, since the number of buddies is much less than points and lots of buddies can be pruned based on the distance bound.

Algorithm 3: Buddy-based Hausdorff distance evaluation

```
   Input: C_n, C_{n+1}
   Output: d_H(C_n, C_{n+1})
1  d_h(C_n, C_{n+1}) ← −∞;
2  L_n ← sort buddies in C_n based on d_max(b, C_{n+1}) in descending
   order;
3  for each buddy b ∈ L_n do
4      if d_max(b, C_{n+1}) ≤ d_h(C_n, C_{n+1}) then
5          Break;
6      m_b ← +∞;
7      L_b ← sort buddies b' in C_{n+1} based on d_min(b, b') in
       ascending order;
8      for each buddy b' ∈ L_b do
9          if d_min(b, b') ≥ d_m then
10             Break;
11         d_m ← the minimum distance between b and b';
12         if d_m ≤ m_b then
13             m_b ← d_m;
14     if m_b > d_h(C_n, C_{n+1}) then
15         d_h(C_n, C_{n+1}) ← m_b;
16 Evaluate d_h(C_{n+1}, C_n) in analogous way;
17 return max{d_h(C_n, C_{n+1}), d_h(C_{n+1}, C_n))};
```

5.3 Incremental Gathering Update

Suppose in the previous phase a crowd $Cr_{old} = \{c_i, \ldots, c_n\}$ in $\mathcal{O}_{DB}$ has been extended into a new closed crowd $Cr_{new} = \{c_i, \ldots, c_n, c_{n+1}\}$ in $\mathcal{O}'_{DB}$. Now our goal is to find new or updated closed gatherings in Cr_{new}. Straightforwardly, we can perform the TAD algorithm on Cr_{new} from scratch, but this approach obviously cannot scale well as the crowds grow longer. To address this problem, we propose an optimization scheme to update the closed gathering patterns incrementally by taking advantage of the gatherings that have already been discovered in Cr_{old}. We will show that this optimization can bring more benefits when Cr_{old} occupies a large portion of Cr_{new}.

As before, the first step is to build the BVS for each object in Cr_{new}. Instead of doing this from scratch, we can simply append another bit indicating the existence of each object in c_{n+1} to the BVS in Cr_{old}, thus saving the time to scan the entire Cr_{new}. Afterwards the *Test*() procedure is invoked to detect the invalid clusters. The following lemma indicates that some original invalid clusters in Cr_{old} may become valid in Cr_{new}.

Lemma 8. *Denote the set of invalid clusters of a crowd Cr as $IC(Cr)$. Then we have $IC(Cr_{new}) \cap Cr_{old} \subseteq IC(Cr_{old})$.*

This is natural since some non-participators in Cr_{old} may turn to be participators because of the new clusters in Cr_{new}. In other words, the gatherings in Cr_{old} may expand or merge with their neighboring gatherings in Cr_{new}. However, if we find some invalid cluster c_j in Cr_{new} which also belongs to Cr_{old}, it is guaranteed that all the closed gatherings before t_j remain unchanged in Cr_{new}. More precisely, we have the following theorem,

Theorem 2. *Given an invalid cluster $c_j \in IC(Cr_{new})$ with $j \leq n+1$, then any closed gathering $Gr \subset \langle c_i, \ldots, c_{j-1} \rangle$ remains closed in Cr_{new}.*

Proof. Since c_j is an invalid cluster, we can only find closed gathering from $Cr_a = \langle c_i, \ldots, c_{j-1} \rangle$ and $Cr_b =$

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
	o_1	o_1		o_1	o_1				o_1	o_1	o_1
o_2	o_2	o_2	o_2			o_2	o_2	o_2	o_2		
o_3	o_3		o_3		o_3	o_3	o_3			o_3	o_3
o_4		o_4	o_4	o_4	o_4	o_4	o_4		o_4	o_4	
	o_5	o_5	o_5						o_5	o_5	o_5
				o_6	o_6			o_6	o_6	o_6	o_6

Fig. 3. Illustration of gatherings update.

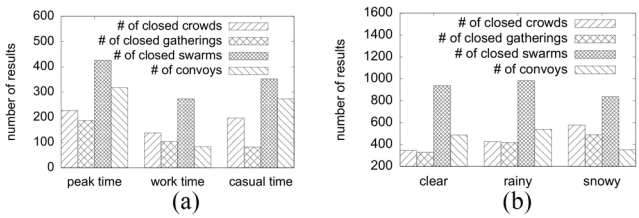

Fig. 4. Effectiveness study.(a) Time of the day. (b) Weather condition.

$\langle c_{j+1}, \ldots, c_m \rangle$. a). If $j = n+1$, Cr_a is actually Cr_{old}. So they have the same set of closed gathering. b). If $j < n+1$, then $c_j \in IC(Cr_{new}) \cap Cr_{old}$. From lemma 8, we know that $c_j \in Cr_{old}$. This means in Cr_{old}, the closed gatherings locate in $Cr_c = \langle c_i, \ldots, c_{j-1} \rangle$ and $Cr_d = \langle c_{j+1}, \ldots, c_n \rangle$. $Cr_a = Cr_c$, hence their closed gatherings are also the same. □

Motivated by Theorem 2, we can improve the original TAD algorithm by utilizing the gatherings found in Cr_{old}. After a set of invalid clusters IC has been obtained in the test phase, we look for the "rightmost" invalid cluster before the timestamp t_{n+1}, i.e., $c_j \in IC(Cr_{new})(j \leq n+1)$, s.t. $\nexists c_{j'} \in IC(Cr_{new})$ that $j < j' \leq n+1$. Theorem 2 guarantees that the closed gatherings on $\langle c_i, \ldots, c_{j-1} \rangle$ remain the same as before, which have already been discovered. Therefore only the sub-crowds within $\langle c_{j+1}, \ldots, c_m \rangle$ need to be examined further since they may contain new or updated gatherings.

Example 3. Continuing with Example 2, the closed crowd in the old database has been extended to a new closed crowd as shown in Fig. 3. Easy to see that all the six objects are participators in the new crowd (recall that o_6 was a non-participator in the old crowd). As a normal TAD algorithm, we perform test on the whole crowd and find c_9 to be an invalid cluster. By Theorem 2, all the closed gatherings in the crowd $\langle c_1, \ldots, c_8 \rangle$ remain closed in the new crowd. So there is no need to test $\langle c_1, \ldots, c_8 \rangle$ again. All we need to do is test the other sub-crowd $\langle c_{10}, c_{11}, c_{12} \rangle$ and validate it as a closed gathering finally.

6 Experiment

In this section, we conduct extensive experiments to evaluate the effectiveness and efficiency of our proposed concepts and algorithms based on a real trajectory dataset, which contains about 120K trajectories generated by over 33,000 taxis of Beijing in a period of 3 months (March, April and May in 2009) [37], [38], [39]. These trajectories are all stored in a database residing on disk. After discretizing the time domain into the granularity of minute, we get 132,480 time points ($60 \times 24 \times 92$) in $\mathcal{T}_{DB}$. Then as an offline preprocessing step, we find the snapshot clusters for every minute by applying the DBSCAN [32] with the settings $m = 5, \epsilon = 200(meters)$. All the clusters are stored in file system. All the algorithms in the following experiments are implemented in C# and run on a computer with Intel Xeon Core 4 CPU (2.66GHz) and 8GB memory.

6.1 Effectiveness

Although the gathering is capable of modelling various group incidents as mentioned in Section 1, in this part we use the traffic condition (e.g.,traffic jams) as a study case to evaluate the effectiveness of our proposals. Intuitively, a traffic jam can be captured by a gathering, since many vehicles with slow speeds form a dense area, and usually most of the vehicles stay within this area for a relatively long time. Essentially, GPS-equipped taxicabs can be viewed as ubiquitous mobile sensors of the city-wide traffic flows. For instance, Beijing has approximately 67,000 licensed taxis generating over 1.2 million occupied trips per day. This figure is around 4.2% of the total personal trips (35 million) within the Six Ring Road of Beijing City (reported by Beijing transportation bureau in July 2010), which is a significant sample reflecting the traffic condition of the city.

In the first experiment, we divide a day into three time periods, peak time (6am to 10am and 5pm – 8pm), work time (10am to 5pm) and casual time (8pm to 5am). Then we find all the closed crowds and gatherings from the trajectory set and group them by the time period with the setting $m_c = 15$, $\delta = 300m$, $k_c = 20$, $k_p = 15$ and $m_p = 10$. As comparison, we also search for the closed swarms and convoys from the trajectories with the settings $min_o = 15, min_t = 10$ (i.e., a group of 15 or more objects travelling together for a period of at least 10 time units). In cases a pattern crosses multiple time periods, we simply duplicate assign it to each of them. Fig. 4(a) shows the average number of each pattern in a single day w.r.t. the time period. It is easy to see that, we can find the most gatherings during the peak time and much fewer for the rest. This observation is consistent with the traffic condition of Beijing, since it experiences severe traffic congestion during the rush hours every day. Interestingly, though there also exist many crowds in casual time, only a small portion turns out to be gatherings. This is because, many crowds are located around restaurants, shopping malls and other entertainment places, where taxicabs usually drop the passengers and leave quickly. As such, these crowds will not form gatherings since there are not enough participators. On the other hand, we can find more swarms and convoys in peak and casual time than in work time. To explain this, many taxicabs have common destination areas during peak (e.g., CBD, residential suburbs) and casual time (e.g., entertainment places). On the contrary, the destinations of most taxicabs are widely distributed during work time, resulting in fewer swarms and convoys.

Next, we categorize the total 92 days into three groups according to the weather condition, namely clear, rainy and snowy. Then we compare the average number of each pattern in a single day with different weather conditions. As shown in Fig. 4(b), we can find the least number of gatherings in clear days and the most in snowy days. The reason is that, as the weather condition becomes worse for the traffic, vehicles tend to move more slowly which makes it easier to cause traffic jams. We also notice the great gap between

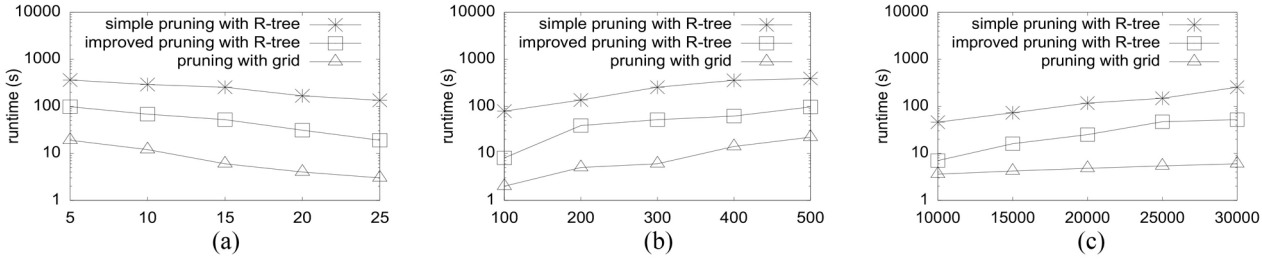

Fig. 5. Time cost of detecting closed crowd in a single day. (a) Running time w.r.t. m_c. (b) Running time w.r.t. δ (meter). (c) Running time w.r.t. $|\mathcal{O}_{DB}|$.

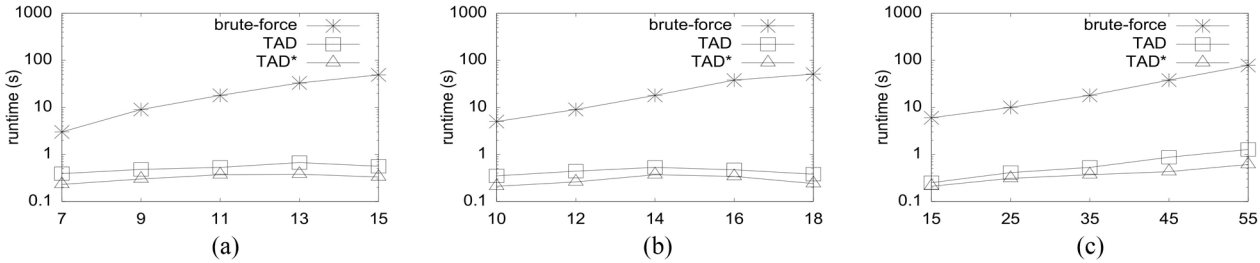

Fig. 6. Running time of closed gathering discovery. (a) Running time w.r.t. m_p. (b) Running time w.r.t. k_p (min). (c) Running time w.r.t. $Cr.\tau$ (min).

the number of crowds and gatherings in snowy days. This may be caused by the large number of minor accidents on the roads, in which most vehicles around the accident can bypass it in a short time. We also note that the number of swarms seems quite insensitive to the weather, while there are fewer convoys in snowy days. A possible explanation is that vehicles try not to travel too closely to each other in snowy weather.

6.2 Efficiency

In this subsection, we study the efficiency of our proposed algorithms. In particular, we will measure the run time of the algorithms for detecting closed crowds, discovering closed gatherings and handling the database update incrementally in different parameter settings.

Performance of crowd detection algorithm. In the first set of experiments, we compare the performances of three pruning schemes in the crowd detection algorithm: a) SR, simple R-tree based pruning with d_{min}; b) IR, improved R-tree based pruning with d_{side}; c) GRID, grid-based pruning. The default parameters used in this set of experiments are: $|\mathcal{O}_{DB}| = 30,000$, $m_c = 15$, $\delta = 300m$, $k_c = 20$. Besides, for the R-tree based method, we set the fanout of R-tree to 50 and page size to 4K; for the grid-based method, we set the side length of each cell to be $\frac{\sqrt{2}}{2} \times \delta$. Below we show the average runtime cost of searching for the closed crowds in a single day, i.e., $|\mathcal{T}_{DB}| = 1440$. It is worth mentioning that, since Algorithm 1 sequentially sweeps all the time points, the parameter k_c only affects the number of gatherings we can find, but has no impact on the time cost of the algorithm. So we omit the experiment studying k_c in the sequel.

As we can see from Fig. 5, IR significantly improves the pruning effect of SR by using a tighter lower bound of d_H. GRID further enhances the performance of IR and outperforms SR by at least one order of magnitude constantly. Specifically, as shown in Fig. 5(a), the runtime costs of all algorithms decrease when m_c increases, since there are less number of clusters satisfying this support threshold at each time instant. As such, there are fewer candidates to consider when we attempt to expand the current crowd candidates. On the contrary, the performances of all methods deteriorate as δ increases (Fig. 5(b)), since the search space increases when we look for the candidate clusters of the next timestamp. Finally, in Fig. 5(c) we study the impact of database size by randomly choosing the subsets of the original dataset with different sizes. As expected, all the schemes need more time to complete on a larger database, since there tends to be more clusters at each time point. Interestingly, however, the grid-based pruning is relatively insensitive to the size of the database. This is due to the fact that, as the affect region of each cluster remains unchanged (since δ remains the same), the refinement cost increases slowly (we only refine the grids in the affect region) even though there may be more clusters considered as candidates.

Performance of gathering discovery algorithm. We evaluate the performances of three algorithms for discovering closed gatherings from a given crowd: a) Brute-force method which will recursively test all the i-length sub-crowds ($i = n, n-1, ...$), until either it finds a gathering or no sub-crowd can be found ($i < k_c$); b) TAD algorithm; c) TAD*: TAD algorithm implemented with the bit vector signature. The default parameters used in this set of experiments are: $m_p = 11$ and $k_p = 14$. For each experiment, we run the algorithms on 1000 closed crowds that are randomly selected and record the average time cost.

From Fig. 6, it is easy to see that TAD outperforms the brute-force method by one to two orders of magnitude, and TAD* further improves TAD by about 30%. In Fig. 6(a), we show the performances of all three algorithms with the variation of m_p, i.e., the least number of participators for a cluster to be valid. As m_p increases, a cluster is more likely to be invalid. For this reason, the brute-force method has to check the shorter sub-crowds with more recursions until it finds a gathering. Although TAD and TAD* also have recursive

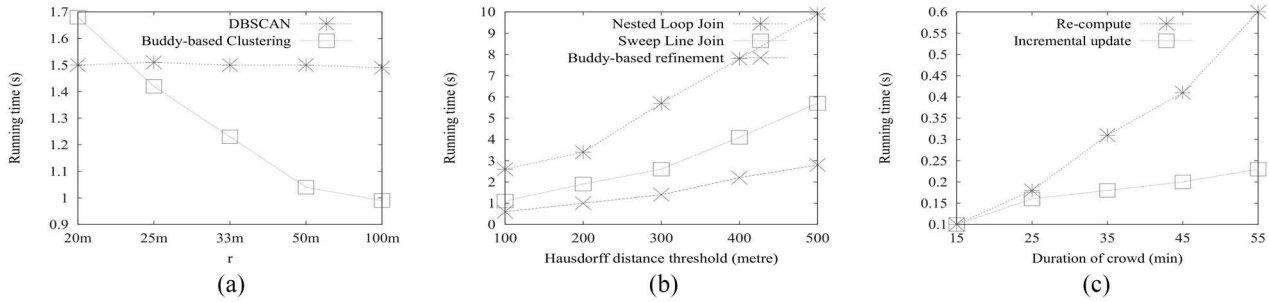

Fig. 7. Performance of gathering monitoring algorithms. (a) Time cost of clustering phase. (b) Time cost of crowd detection for a new time instance. (c) Time cost of gathering discovery in a single crowd.

procedures, they do not enumerate all the subsequences of a crowd. Interestingly, with the further increase of m_p, the time costs of TAD and TAD* turn to decrease. This is because too many invalid clusters in the original crowd will result in a large number of subsequences that are non-crowd, hence making the recursion terminate more quickly. Fig. 6(b) shows the effect of the other parameter k_p, which is the least time period for a participator to stay within the crowd. As the previous experiment, there will be less valid clusters with greater k_p since the number of participators decreases. We omit the analysis for k_p due to its similarity with m_p.

We also investigate the runtime cost when the algorithms are performed on the crowds with different lengths ($Cr.\tau$), the results of which are shown in Fig. 6(c). As expected, the time cost of the brute-force method increases almost exponentially with $Cr.\tau$, since the number of subsequences is exponential to the length of a crowd. The performances of the other two algorithms also deteriorate with the length of the crowd, but the changes are more smooth. In addition, TAD* exhibits more benefits on longer $Cr.\tau$, since using the BVSs for a longer sequence can save more computation time.

Performance of monitoring algorithms. Finally we analyse the performance of the proposed monitoring algorithms for handling streaming trajectories. To simulate the streaming environment, we continuously append the trajectories in each time instant to the existing dataset and try to update the gathering patterns immediately. Fig. 7 shows the time cost of different algorithms for the three phases, namely snapshot clustering, crowd detection and gathering discovery, respectively. First, we examine how the threshold of buddy radius affects the performance of the buddy-based clustering. As shown in Fig. 7(a), when the radius threshold is very small (20m), buddy-based clustering algorithm is even more costly than DBSCAN. This is because a large number of buddies needs to be modified at such a small threshold that incurs high runtime overhead. As the radius threshold gradually increases, the efficiency of buddy-based clustering algorithm gets improved since many buddies remain the same over consecutive time instants. According to this experiment, we set the radius threshold to be 100m. It is worth to mention that, the choice of radius threshold will not affect the clustering results. It can only affect the size and number of buddies and hence the performance of the clustering algorithm. Second, we compare the efficiency of different on-line crowd detection approaches and report the average time cost for detecting all the crowds in a new time instant. From Fig. 7(b) we can see that using sweep line algorithm to find candidate clusters in the pruning step is significantly more efficient than the nested loop join approach. Besides, the buddy-based refinement optimization can further speed up the crowd detection process by using the inter-buddy distance to reduce unnecessary distance computations. We also notice that the performance benefit becomes more significant when the threshold of Hausdorff distance increases. Last, we test the performance of incremental gathering update algorithm by comparing it against the re-computation method (e.g., re-invoke the TAD* procedure whenever the crowd gets updated). Fig. 7(c) shows the average time cost of discovering the gatherings on a single crowd with respect to the duration of the crowd. Not surprisingly, the runtime of re-computation approach increases quickly with the duration of the crowd, therefore it is not suitable for the on-line monitoring scenarios where the crowds keep growing. On the contrary, the incremental update approach can scale much better with the length of crowd as it leverages the old gatherings that was previously discovered in the crowd to eliminate lots of re-discovery.

7 Conclusion

In this paper, we study the problem of online discovering gathering patterns over large-scale, dynamic trajectory databases. Different from the earlier proposed concepts, such as flock, convoy and swarm, which aim to identify groups of moving objects travelling together for a certain time period, the gatherings are able to model a variety of non-trivial group events or incidents. Since the whole discovery process could be very time consuming in a large trajectory dataset, we propose a set of techniques to improve the efficiency. Besides, we also develop an efficient online monitoring solution that can keep track of pattern updates while new trajectory data are coming. Extensive experiments based on real taxicab trajectory dataset have demonstrated that the proposed online discovery algorithms can maintain the patterns for whole dataset within a few seconds, which is efficient enough for many practical applications.

ACKNOWLEDGMENTS

This research is partially supported by Natural Science Foundation of China (Grant 61232006) and the Australian Research Council (Grants DE140100215, DP110103423 and DP120102829).

REFERENCES

[1] Y. Zheng and X. Zhou, *Computing with Spatial Trajectories*. New York, NY, USA: Springer, 2011.
[2] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. Tsotras, "On-line discovery of dense areas in spatio-temporal databases," in *Advances in Spatial and Temporal Databases*. Berlin, Germany: Springer, 2003, pp. 306–324.
[3] C. Jensen, D. Lin, B. Ooi, and R. Zhang, "Effective density queries on continuously moving objects," in *Proc. ICDE*, 2006, p. 71.
[4] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle, "Reporting flock patterns," *Comput. Geom.*, vol. 41, no. 3, pp. 111–125, Nov. 2008.
[5] M. Vieira, P. Bakalov, and V. Tsotras, "On-line discovery of flock patterns in spatio-temporal data," in *Proc. ACM GIS*, Sydney, NSW, Australia, 2009, pp. 286–295.
[6] J. Gudmundsson and M. van Kreveld, "Computing longest duration flocks in trajectory data," in *Proc. ACM GIS*, Arlington, VA, USA, 2006, pp. 35–42.
[7] J. Gudmundsson, M. van Kreveld, and B. Speckmann, "Efficient detection of motion patterns in spatio-temporal data sets," in *Proc. ACM GIS*, Washington, DC, USA, 2004, pp. 250–257.
[8] G. Al-Naymat, S. Chawla, and J. Gudmundsson, "Dimensionality reduction for long duration and complex spatio-temporal queries," in *Proc. ACM Symp. Applied Computing*, Seoul, Korea, 2007, pp. 393–397.
[9] H. Jeung, M. Yiu, X. Zhou, C. Jensen, and H. Shen, "Discovery of convoys in trajectory databases," *VLDB Endow.*, vol. 1, no. 1, pp. 1068–1080, Aug. 2008.
[10] H. Jeung, H. Shen, and X. Zhou, "Convoy queries in spatio-temporal databases," in *Proc. ICDE*, Cancun, Mexico, 2008, pp. 1457–1459.
[11] Z. Li, B. Ding, J. Han, and R. Kays, "Swarm: Mining relaxed temporal moving object clusters," *VLDB Endow.*, vol. 3, no. 1, pp. 723–734, Sep. 2010.
[12] G. Rote, "Computing the minimum Hausdorff distance between two point sets on a line under translation," *Inform. Process. Lett.*, vol. 38, no. 3, pp. 123–127, 1991.
[13] P. Kalnis, N. Mamoulis, and S. Bakiras, "On discovering moving clusters in spatio-temporal data," in *Advances in Spatial and Temporal Databases*. Berlin, Germany: Springer, 2005, pp. 364–381.
[14] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang, "On discovery of gathering patterns from trajectories," in *Proc. ICDE*, Brisbane, QLD, Australia, 2013, pp. 242–253.
[15] S. Gaffney, A. Robertson, P. Smyth, S. Camargo, and M. Ghil, "Probabilistic clustering of extratropical cyclones using regression mixture models," *Clim. Dyn.*, vol. 29, no. 4, pp. 423–440, Sep. 2007.
[16] S. Gaffney and P. Smyth, "Trajectory clustering with mixtures of regression models," in *Proc. SIGKDD*, New York, NY, USA, 1999, pp. 63–72.
[17] J. Lee, J. Han, and K. Whang, "Trajectory clustering: A partition-and-group framework," in *Proc. SIGMOD*, Beijing, China, 2007, p. 604.
[18] Z. Li, J. Lee, X. Li, and J. Han, "Incremental clustering for trajectories," in *Proc. Database Systems Advanced Applications*, Berlin, Germany, 2010, pp. 32–46.
[19] N. Mamoulis *et al.*, "Mining, indexing, and querying historical spatiotemporal data," in *Proc. SIGKDD*, Washington, DC, USA, 2004, pp. 236–245.
[20] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi, "Trajectory pattern mining," in *Proc. SIGKDD*, New York, NY, USA, 2007, pp. 330–339.
[21] Y. Liu, L. Chen, J. Pei, Q. Chen, and Y. Zhao, "Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays," in *Proc. IEEE Int. Conf. Pervasive Computing Communications*, White Plains, NY, USA, 2007, pp. 37–46.
[22] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. Sondag, "Adaptive fastest path computation on a road network: A traffic mining approach," in *Proc. VLDB*, Vienna, Austria, 2007, pp. 794–805.
[23] Y. Zheng, L. Zhang, X. Xie, and W. Ma, "Mining interesting locations and travel sequences from gps trajectories," in *Proc. WWW*, Madrid, Spain, 2009, pp. 791–800.
[24] A. Y. Xue *et al.*, "Destination prediction by sub-trajectory synthesis and privacy protection against such prediction," in *Proc. ICDE*, Brisbane, QLD, Australia, 2013, pp. 254–265.
[25] E. Knorr, R. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *VLDB J.*, vol. 8, no. 3, pp. 237–253, Feb. 2000.
[26] X. Li, J. Han, S. Kim, and H. Gonzalez, "ROAM: Rule-and motif-based anomaly detection in massive moving object data sets," in *Proc. SDM*, 2007.
[27] J. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *Proc. ICDE*, Cancun, Mexico, 2008, pp. 140–149.
[28] Y. Bu, L. Chen, A. Fu, and D. Liu, "Efficient anomaly monitoring over moving object trajectory streams," in *Proc. SIGKDD*, Paris, France, 2009, pp. 159–168.
[29] W. Wang, J. Yang, and R. Muntz, "Sting: A statistical information grid approach to spatial data mining," in *Proc. VLDB*, Athens, Greece, 1997, pp. 186–195.
[30] D. Agarwal, A. McGregor, J. Phillips, S. Venkatasubramanian, and Z. Zhu, "Spatial scan statistics: Approximations and performance study," in *Proc. SIGKDD*, Philadelphia, PA, USA, 2006, pp. 24–33.
[31] M. Kulldorff, "A spatial scan statistic," *Commun. Statist. Theory Methods*, vol. 26, no. 6, pp. 1481–1496, 1997.
[32] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. SIGKDD*, vol. 96. 1996, pp. 226–231.
[33] D. Knuth, *The Art of Computer Programming, Fascicle 1: Bitwise Tricks and Techniques; Binary Decision Diagrams*, vol. 4. Boston, MA, USA: Addison-Wesley, 2009.
[34] L. Tang *et al.*, "On discovery of traveling companions from streaming trajectories," in *Proc. ICDE*, Washington, DC, USA, 2012, pp. 186–197.
[35] P. Mishra and M. H. Eich, "Join processing in relational databases," *ACM Comput. Surv.*, vol. 24, no. 1, pp. 63–113, 1992.
[36] F. Preparatat and M. I. Shamos, *Computational Geometry: An Introduction*. New York, NY, USA: Springer-Verlag, 1985.
[37] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. SIGKDD*, New York, NY, USA, 2011, pp. 316–324.
[38] J. Yuan *et al.*, "T-drive: Driving directions based on taxi trajectories," in *Proc. GIS*, New York, NY, USA, 2010, pp. 99–108.
[39] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *Proc. ICDE*, Washington, DC, USA, 2012.

Kai Zheng is an ARC DECRA (Australia Research Council Discovery Early-Career Researcher Award) Fellow with The University of Queensland. He obtained his Ph.D degree in Computer Science from The University of Queensland in 2012. His research interests include uncertain database, spatial-temporal query processing, and trajectory computing. He is a member of the IEEE.

Yu Zheng is a lead researcher at Microsoft Research and a Chair Professor at Shanghai Jiaotong University. He joined MSRA in July 2006 right after received his Ph.D. degree in communication & information system from Southwest Jiaotong University. His research interests include location-based services, spatio-temporal data mining, and ubiquitous computing. He was named Top Innovator Under 35 (TR35) by MIT Technology Review in 2013 for his research using big data to solve urban challenges. He is a senior member of the IEEE.

Nicholas J. Yuan is an associate researcher at Microsoft Research. He acquired his Ph.D degree in Computer Science in 2012 and his B.S degree in Mathematics in 2007, both from University of Science and Technology of China. His research interests include spatial-temporal data mining, behavioral mining and computational social science.

Shuo Shang is a Research Professor at China University of Petroleum. He was a research assistant professor at Aalborg University. His research interests include Efficient Query Processing in Spatio-Temporal Databases, Trajectory Search and Mining.

Xiaofang Zhou is a Professor of computer science at The University of Queensland and Adjunct Professor in the School of Computer Science and Technology, Soochow University, China. He received the B.Sc. and M.Sc. degrees in computer science from Nanjing University, China, in 1984 and 1987, respectively, and the Ph.D. degree in computer science from The University of Queensland, Australia, in 1994. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, bioinformatics, and e-research. He is a senior member of the IEEE.