# Detecting Moving Object Outliers In Massive-Scale Trajectory Streams

Yanwei Yu *
Yantai University
Yantai, Shandong China
yuyanwei0530@126.com

Lei Cao *
Elke A. Rundensteiner
Worcester Polytechnic Institute
Worcester, MA U.S.A
lcao|rundenst@cs.wpi.edu

Qin Wang
University of Science and Technology Beijing
Beijing, China
wangqin@ies.ustb.edu.cn

## ABSTRACT

The detection of abnormal moving objects over high-volume trajectory streams is critical for real time applications ranging from military surveillance to transportation management. Yet this problem remains largely unexplored. In this work, we first propose classes of novel trajectory outlier definitions that model the anomalous behavior of moving objects for a large range of real time applications. Our theoretical analysis and empirical study on the Beijing Taxi and GMTI (Ground Moving Target Indicator) datasets demonstrate its effectiveness in capturing abnormal moving objects. Furthermore we propose a general strategy for efficiently detecting the new outlier classes. It features three fundamental optimization principles designed to minimize the detection costs. Our comprehensive experimental studies demonstrate that our proposed strategy drives the detection costs 100-fold down into practical realm for applications producing high volume trajectory streams to utilize.

## Categories and Subject Descriptors

H.2 [**Information Systems**]: Database Management

## Keywords

Outlier, Moving Object, Trajectory Stream

## 1. INTRODUCTION

**Motivation.** In recent years, the location-acquisition devices like GPS, smart phone, and RFID have become prevalent. These devices, monitoring the motion of vehicles, people, goods, services, and animals, are producing massive-volume high-speed trajectory streams. Many applications from traffic management [5], security surveillance [2], scientific studies [13], to mobile social networks [18] rely on continuously discovering abnormal objects in such trajectory streams to deliver critical decisions within an actionable time.

*Authors contribute equally to this work. This work was done when Yanwei Yu visited Worcester Polytechnic Institute.

In security surveillance systems, a visitor at a miliary base will be considered as an outlier and thus a potential safety threat if he does not obey the strict order to stay together with his designated group members. In traffic management systems a taxi driver will be classified as an outlier in terms of his erratic behavior if he keeps changing lanes and switching his neighboring travel companions. Potentially this may help flag speeding, drunk driving, or other erratic behaviors of concern.

Similarly, if considering the price, volume, or gains of stocks at a particular time point as coordinates in a multi-dimensional space, then real time stock quotes can be modeled as a trajectory stream. Its analysts may reveal recent promising stocks by detecting the outlier stocks whose performance trajectories dramatically deviate from those of other stocks in the same industry.

**Challenges.** In all the applications described above, outliers can be characterized as moving objects that behave differently from the majority in trajectory streams. Despite the importance of continuously detecting such types of outliers, to the best of our knowledge, this problem has not previously been considered in the literature.

*In the streaming context* Bu et al. [2] defined when to consider a trajectory *segment* of a single moving object to be an outlier, while our work instead focuses on a much more complicated problem, namely locating outlier objects in the trajectory stream populated with massive scale moving objects. In [2] a given trajectory is divided into equal sized segments. A segment is said to be abnormal if it is not similar to a group of segments adjacent to it in time. The effectiveness of this definition is based on the *local continuity* observation. Namely one moving object usually is expected to behave consistently within a short time interval. However in our context whether one moving object is an outlier or not depends on its relationship with other objects. The moving patterns of a large set of objects are more complex and dynamic than one single object's path and hence cannot be modeled by the local continuity property. Therefore this definition cannot be applied to our problem.

*In static spatio-temporal (time series) databases*, a trajectory is said to be an outlier if it does not show the global characteristics of the majority of the trajectories in the overall database [8, 11]. Since all trajectories are known apriori, these techniques rely on expensive offline pre-processing. Normally they first mine all frequent patterns to build a model of global characteristics of the dataset. In the second phase the model is then used to classify each trajectory as being either an outlier or not. However in unbounded continuous trajectory streams where concept drift consistently arises, using one single (pre-computed) model to continuously detect outliers would inevitably lead to inaccurate results, while periodically rebuilding the global model can be prohibitively expensive for real time stream trajectory outlier detection due to the modeling costs.

Therefore to effectively identify abnormal objects in trajectory streams, new semantics have to be defined to satisfy the requirement of streaming trajectory outlier detection. First, lightweight metrics suitable to identify outliers in place of complicated statistical models are desirable. These metrics must capture the key characteristics of moving objects. Second, given the dynamic nature of stream trajectories this definition should be robust to concept drift and amenable to swiftly evicting obsolete models of outlierness.

Furthermore, if such a lightweight yet effective outlier definition could be found, we must be able to process this detection semantics to assure we can extract real time insights from *high volume* trajectory stream data, such as the streams of surveillance, stocks, or traffic. Thus strategies must be designed to efficiently discover the stream trajectory outliers online as stream data passes by.

**Proposed Solution.** In this paper, we propose new trajectory outlier definitions by introducing the notion of "trajectory neighbor" to measure the similarity among different trajectories. Unlike the traditional neighbor definition [8] which simply considers the physical distance among two objects at a given point of time, the trajectory neighbor concept captures the key properties of stream trajectories. Namely it not only considers the spatial proximity of trajectory objects, but also takes the duration of the spatial similarity across time into account.

Furthermore our analysis of realtime trajectory applications reveals that applications vary in their particular synchronization requirements with respect to the neighbor relationships among multiple trajectories. We thus propose two variants of the trajectory neighbor concept with different synchronization regulations that facilitate users to customize the semantics via controlled parameters. By measuring the *Precision* and *Recall* on the *GMTI* [3] and *Taxi* [16, 17] data, our empirical study confirms that the new definitions successfully model the deviating behavior that characterizes outliers in a rich variety of trajectory stream applications.

Moreover we design a comprehensive strategy to efficiently detect these new outlier types over high volume trajectory streams, called the *minimal examination* (MEX) framework.

The MEX framework integrates three fundamental optimization principles that apply to both neighbor-based outlier definitions. First, given a trajectory $Tr_i$, the *minimal support examination* principle guides MEX to always acquire only the minimal yet sufficient set of neighbor evidence called *minimal support*. The key insight is to stop immediately once the outlier status of $Tr_i$ has been proven. Second, MEX leverages the temporal relationships among trajectory points to prioritize the processing order among points during the neighbor search process. This principle, called *time-aware examination*, guarantees that we find the relationships most useful for outlier detection. Furthermore MEX establishes the *lifetime* concept signaling the furthest window up to which we can predict the status of $Tr_i$ based on current trajectory evidence. This enables MEX to transform the periodical per-window based outlier detection process into a *lifetime-triggered* detection process. Our experimental studies on the Taxi data show that MEX can successfully handle up to one million moving objects per second on a standard desktop, rendering trajectory outlier detection practical in massive-scale moving object streams.

**Contributions.** The main contributions of this paper include: 1) We propose novel neighbor-based trajectory outlier definitions which are theoretically shown to successfully model the abnormal behavior of moving objects in the trajectory stream. 2) We design incremental algorithms for trajectory outlier detection which leverage the overlap of sliding windows. Our complexity analysis highlights the need for more sophisticated techniques. 3) We thus propose the *minimal examination* (MEX) framework equipped with three core optimization principles. 4) We propose efficient algorithms for the new trajectory outlier definitions based on the MEX framework, successfully driving the detection costs 100-fold down. 5) Our empirical study using GMTI and Taxi datasets demonstrate the robust *Precision* and *Recall* of our new proposed definitions.

## 2. NEIGHBOR-BASED TRAJECTORY OUTLIER DETECTION

### 2.1 Notations

We denote the set of $n$ moving objects as $MO = \{o_1, o_2, \ldots, o_n\}$, where $o_i$ is the moving object with $id = i$. The multi-dimensional data point $p_i^j$ produced by the moving object $o_i$ at time $t_j$ is called a trajectory point of $Tr_i$. We assume a minimal time interval at which objects produce events as trajectory points. We utilize the term "timebin" to refer to this smallest time granularity. The trajectory of a moving object $o_i$ is thus an infinite sequence of trajectory points produced at timebins $\{t_1, t_2, \ldots, t_j, \ldots\}$, denoted as $Tr_i$=$\{p_i^1, p_i^2, \ldots p_i^j, \ldots\}$. In this work a moving object is not limited to an object equipped with location-acquisition device. Instead it is an abstraction of any object that continuously observes distinct events along time. Correspondingly the trajectory point is not necessarily only a spatial position, but could be any multi-dimensional coordinate where each dimension corresponds to one of the domains of the attributes of an object's observation.

We define a trajectory stream $S$ of $n$ moving objects in $MO$ as an infinite sequence of trajectory points ordered by timebins $S$ = $\{p_1^1 p_2^1 \ldots p_n^1,\ p_1^2 p_2^2 \ldots p_n^2,\ \ldots,\ p_1^i p_2^i \ldots p_n^i,\ \ldots\}$. Trajectory points $p_1^i p_2^i \ldots p_n^i$ are said to fall into the same timebin $i$ in the stream $S$.

In this work, we use the periodic sliding window semantics as proposed by CQL [1] to define a finite sub-stream of an infinite trajectory data stream. In particular, a query $Q$ specifies a fixed window size $w$ and slide size $s$ for time-based windows, while count-based windows can similarly be defined. Each window $W$ has a starting time $W.T_{start}$ and an ending time $W.T_{end} = W.T_{start} + w-1$. The population of the current window $W_c$ of $S$ consists of all points whose timebin falls into $W_c$. It is the finite subsequence of the trajectory stream $S$: $\{p_1^{t_c-w+1} p_2^{t_c-w+1} \ldots p_n^{t_c-w+1}, p_1^{t_c-w+2} p_2^{t_c-w+2} \ldots p_n^{t_c-w+2}, \ldots, p_1^{t_c} p_2^{t_c} \ldots p_n^{t_c}\}$, where $t_c$ is the current timebin. Periodically the current window $W_c$ slides, causing $W.T_{start}$ and $W.T_{end}$ to increase by $s$ timebins. Henceforth a trajectory $Tr_i$ refers to a set of trajectory points produced by one moving object $o_i$ throughout a window $W_j$.

We use the function $dist(p_m^j, p_n^j)$ to denote the distance between two trajectory points $p_m^j$ and $p_n^j$ at the same timebin $j$. Without loss of generality, we utilize Euclidean distance as the distance measure, though any other distance measure could equally be plugged in.

**Definition** 1. ***(Point Neighbor).*** *For two trajectory points $p_m^j$ and $p_n^j$ in the same timebin $t_j$, if $dist(p_m^j, p_n^j) \le d$, we say that $p_m^j$ is a **point neighbor** of $p_n^j$, with $d$ a given distance threshold.*

For point $p_n^j$, all point neighbors of $p_n^j$ at timebin $t_j$ w.r.t. the distance threshold $d$ comprise the **Point Neighbor Set** of $p_n^j$, denoted as $PN(p_n^j, d)$.

**Definition** 2. ***(Trajectory Neighbor).*** *In window $W$, trajectory $Tr_m$ is called a **trajectory neighbor** of $Tr_n$ w.r.t. the given timebin count threshold $thr_t$, iff there exist at least $thr_t$ timebins in W such that $p_m$ is a point neighbor of $p_n$ at each of these $thr_t$ timebins with point neighbor as per Def. 1.*

The set of trajectory neighbors of a trajectory $Tr_n$ in the window $W_c$, w.r.t. distance threshold $d$ and timebin count threshold $thr_t$ is denoted as $TN(Tr_n, d, thr_t)$.

## 2.2 Trajectory Outlier Definitions

### 2.2.1 Point-Neighbor Based Trajectory Outlier

**Definition** 3. *Given a distance threshold d, a neighbor count threshold k, and timebin count threshold $thr_t$, a trajectory $Tr_i$ in the window W is called a* ***point-neighbor based trajectory outlier****, or in short,* ***PN-Outlier****, if $|T| < thr_t$, where $T = \{t_j | \, |PN(Tr_i, t_j, d)| \geq k, \; t_j \in [W.T_{Start}, W.T_{End}]\}$. Otherwise $Tr_i$ is a* ***PN-Inlier****. Any timebin $t_j$ with $|PN(p_i^j, d)| \geq k$ is called a neighboring timebin of $Tr_i$.*

This *PN-Outlier* is based on whether a trajectory has a sufficient number of neighboring timebins throughout the window. Intuitively, a normal inlier trajectory has a sufficient number of moving objects in its vicinity for most of the time, i.e., for at least $thr_t$ timebins. That is, such inliers are in a crowd in at least $thr_t$ timebins. On the other hand, a trajectory is said to be an *outlier* if most of the time it differs from other objects in a significant fashion.

**Example** 1. *See example of Def. 3 in Fig. 1. Suppose $k = 2$, $thr_t = 4$. The trajectory $Tr_3$ only has two neighboring timebins $\{t_1, \; t_4\}$, for which $Tr_3$ has more than k point neighbors. Thus $|T| < 4$. So $Tr_3$ is a PN-Outlier. All other trajectories have at least $thr_t$ neighboring timebins, hence are PN-Inliers.*

**Applications.** *PN-Outlier* effectively models the notion of outliers prevalent in many real time applications. For example in the security surveillance, foreign personnel may be considered as potentially unsafe if he separates out from others for too much of the time during his visit. In a marathon race a runner who runs alone without any other competitors around him for long stretches of time might be considered as suffering from some physical discomfort and possibly needing medical assistance, or should be checked for possible cheating by taking shortcuts.

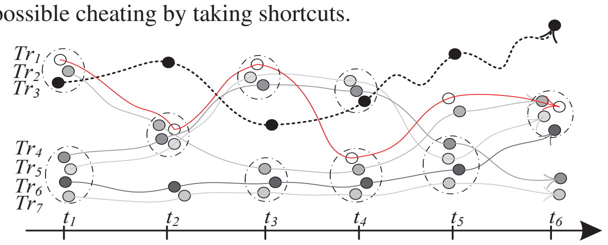


**Figure 1: 7 trajectories in a window $W_c$ with size $w = 6$. Small circle represents a trajectory point. Dashed oval indicates pairwise point neighbors.** $PN$-$Outlier$: $Tr_3$. $PN$-$Inliers$: $Tr_1$, $Tr_2$, $Tr_4$, $Tr_5$, $Tr_6$, **and** $Tr_7$.

### 2.2.2 Trajectory-Neighbor Based Trajectory Outlier

**Definition** 4. *Given a distance threshold d, a neighbor count threshold k, and timebin count threshold $thr_t$, a trajectory $Tr_i$ in the window W is a* ***trajectory-neighbor based trajectory outlier****, or in short,* ***TN-Outlier****, if $|TN(Tr_i, d, thr_t)| < k$ in W. Otherwise the trajectory $Tr_i$ is a* ***TN-Inlier****.*

This *TN-Outlier* definition classifies a trajectory $Tr_i$ as outlier if it does not have a sufficient number of trajectory neighbors throughout the window $W$. Put differently, there are not enough other objects in the dataset that consistently behave similarly to $Tr_i$ within the observed time period $W$.

**Example** 2. *Consider the trajectories in Fig. 2. Suppose $k = 2$, $thr_t = 4$. Then $Tr_4$ is a TN-Outlier having no trajectory neighbor. $Tr_1$ and $Tr_2$ are trajectory neighbors of each other by Def. 2, because they are point neighbors at all 6 timebins. $Tr_1$, $Tr_2$, and $Tr_7$ are pairwise trajectory neighbors, because all three trajectories meet at at least 4 timebins. Since $Tr_1$, $Tr_2$ and $Tr_7$ each have k (2) trajectory neighbors, they are TN-Inliers. $Tr_3$ has two trajectory neighbors $Tr_5$ and $Tr_6$. Thus $Tr_3$ is a TN-Inlier.*

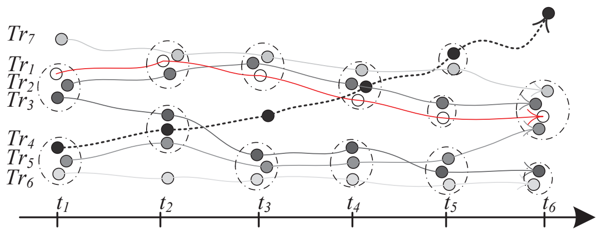


**Figure 2: 7 trajectories in a window with size *w*=6, *k* = 2, $thr_t$ = 4. *TN-Outlier*: $Tr_4$; *TN-Inlier*: $Tr_1$, $Tr_2$, $Tr_3$, $Tr_5$, $Tr_6$, $Tr_7$.**

Unlike the *PN-Outlier* definition (Def. 3), *TN-Outlier* concept does not care about whether a given trajectory has a sufficient number of neighboring timebins, rather the granularity of similarity is instead at the level of complete trajectories and their relationship with $Tr_i$. Thus although $Tr_4$ has four neighboring timebins ($\geq thr_t$), it is still a *TN-Outlier* as we have shown above.

**Applications.** *TN-Outlier* definition fits many real life applications. Consider traffic management applications [15] where we expect most drivers to drive consistently in lockstep with neighboring cars on a highway, for instance in the same or in adjacent lanes. Deviating from the majority of other cars may indicate that the drivers change their neighbors frequently due to speeding (too fast) or drunk driving (too slow) and thus never stay long enough at similar speed with other cars. Similarly in the stock sticker stream, stocks in the same industry tend to exhibit similar trends. Therefore a stock whose performance consistently deviates from that of other stocks in the same industry will be considered as a *TN-Outlier*, even if at many timebins some stocks (each time different ones) happen to be its neighbors.

### 2.2.3 Effectiveness Analysis

Now we analyze the effectiveness of our proposed definitions. Due to space restriction in this work we only focus on *PN-Outlier*. A similar analysis could be applied to *TN-Outlier*.

Although no single outlier definition effectively models all real world scenarios, the general notion of what constitutes an outlier was first introduced by Hawkins in [7]. An observation is said to be outlier if it statistically deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism.

Next we establish a statistical model for trajectory outliers. It relies on the strong assumption that the stream data consistently follows a particular distribution. In reality we may not know the distribution of a given stream. Worst yet it may change its distribution at any moment, while computing the statistical model periodically would be highly inefficient. However idealistically it still provides a solid baseline to contrast our definition that rests upon zero assumption.

To facilitate analysis, let us consider the scenario where each trajectory point set $D_i$ composed of all trajectory points at any timebin $t_i$ follows the *normal distribution* $N(\mu, \sigma^2)$. A trajectory point $Tr_j^i$ is considered as an outlier point in $D_i$ if it lies 3 or more standard deviation $\sigma$ from the mean $\mu$ [4]. Correspondingly given a trajectory $Tr_j$ with $W$ trajectory points in a window, the probability of $Tr_j$ having *m* of $W$ outlier trajectory points will follow the binomial distribution $Bin(m|W, \mu')$ with $\mu'$ representing the probability of a given trajectory point $Tr_j^i$ to be an outlier point in $D_i$. Then statistically trajectories can be considered as outliers if they contain more than $W\mu' + 3\sqrt{W\mu'(1-\mu')}$ outlier points. Here $W\mu'$ is the expectation of *Bin*, while $W\mu'(1-\mu')$ is the variance of *Bin*. We denote this definition as $Def_{stat}$.

Now we show the effectiveness of *PN-Outlier* by proving the following lemma.

**Lemma** 1. *Trajectory $Tr_j$ is an outlier according to $Def_{stat}$ iff $Tr_j$ is a PN-Outlier with d = $0.13\sigma$, k = 0.0012W, $thr_t = 0.9972W - 3\sqrt{0.0028(1-0.0028)W}$.*

Lemma 1 shows that our lightweight *PN-Outlier* definition will indeed produce the identical outliers that $Def_{stat}$ had produced if the stream is normal distributed for a given set of statistical determined parameters.

PROOF. We use probabilities to represent the number of points lying in a d-neighborhood. Specifically, the probability $p$ is $\frac{k}{W} = 0.0012$ that the distance between 2 randomly sampled points $s_1$ and $s_2$ from $D_i$ is less than or equal to d. $s_1$ and $s_2$ can be considered to be random variables following normal distribution $N(\mu, \sigma^2)$. Then we define $z_1 = \frac{s_1 - \mu}{\sigma}$ and $z_2 = \frac{s_2 - \mu}{\sigma}$ as standard normal variables. That is, $z_1, z_2 \sim N(0, 1)$. Given d = $0.13\sigma$, $p$ = 0.0012, we get:

$Pr(|s_1 - s_2| \leq 0.13\sigma) \leq 0.0012 \Leftrightarrow Pr(s_1 - 0.13\sigma \leq s_2 \leq s_1 + 0.13\sigma) \leq 0.0012 \Leftrightarrow Pr(z_1 - 0.13 \leq z_2 \leq z_1 + 0.13) \leq 0.0012$.

Using the normal distribution table [4] we get $z_1 \leq -3.0$ or $z_1 \geq 3.0 \Leftrightarrow s_1 \leq \mu - 3\sigma$ or $s_1 \leq \mu + 3\sigma$. This proves that an outlier timebin $t_i$ of trajectory $Tr_j$ in *PN-Outlier* definition is an outlier point of $D_i$ in $Def_{stat}$.

Next, by $Def_{stat}$ the probability of $s_1$ being an outlier point will be $Pr(z_1 \leq -3.0) + Pr(z_1 \geq 3.0) = 2Pr(z_1 \geq 3.0) = 2(1 - Pr(z_1 \leq 3.0)) = 0.0028$. In other words, the mean of *Bin* $\mu'$ = 0.0028. Therefore the expectation of *Bin* $E[m] = 0.0028W$. Its variance $var[m] = 0.0028(1-0.0028)W$. By $Def_{stat}$ trajectory $Tr_j$ will be an outlier if it has more than $E[m] + 3\sqrt{var[m]}$ outlier points. Since $E[m] + 3\sqrt{var[m]}$ = $W - thr_t$, this is equivalent to the condition of *PN-Outlier* with $thr_t$ as the minimum number of neighboring timebins for $Tr_j$ to be an inlier. □

## 3. THE BASIC OUTLIER DETECTOR

Given a trajectory stream $S$ and the semantics of *PN-Outlier* or *TN-Outlier*, we need to design algorithms to continuously detect and output the trajectory outliers in each window $W_c$. For this we now design the INC algorithm which detects both classes of outliers in a unified way. It first utilizes range queries to search for all point neighbors of each trajectory $Tr_i$ at each timebin. Then the acquired point neighbors are used as evidence to validate the status of $Tr_i$. Furthermore INC leverages the fact that in sliding window streams adjacent windows overlap with each other. By cleverly maintaining the already known neighbor relationships, the INC algorithm successfully eliminates any redundant range query search at the previously examined timebins, resulting in saving in system resources.

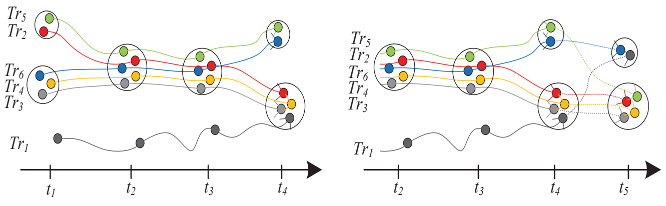
We first introduce the data structures that enable the INC algorithm to extensively reuse the already recognized neighbor-related information. We use $DB_{Tr}$ to denote the set of all trajectories in the current window $W_c$. To support *PN-Outlier* detection, each trajectory $Tr_i$ maintains a list $Tr_i.tlist$, which stores the IDs of its neighboring timebins in $W_c$. For *TN-Outlier* detection, we need to track the trajectory neighbors of $Tr_i$ to evaluate whether $Tr_i$ has a sufficient number of trajectory neighbors. Therefore a neighbor table, denoted as $Tr_i.NT$, is maintained by each trajectory $Tr_i$, which records information about all trajectories having at least one point neighbor with $Tr_i$ in the current window (see Fig. 4). Each record in the neighbor table $Tr_i.NT$ is a $< key, valueList >$ pair, where *key* denotes the identifier of $Tr_i$'s neighboring trajectory $Tr_j$, and *valueList* corresponds to the list of timebins during which $Tr_i$ and $Tr_j$ are point neighbors.

Using an example-driven approach, we now describe how the INC algorithm detects the two outlier classes with the assistance of the just introduced $Tr_i.tlist$ and $Tr_i.NT$ structures. Fig. 3 shows 6 trajectories in two consecutive sliding windows with window size $w = 4$ and slide size $s = 1$. Given a query $Q$ with $k = 2$ and $thr_t = 3$. Fig. 4 illustrates how the INC algorithm incrementally processes $Tr_1$ and $Tr_5$.

**Data Structure Initialization.** Given the trajectory dataset in the first window $W_1$, INC first utilizes a range query operation to search for point neighbors of $Tr_1$ at timebin $t_1$. The distance threshold *d* is used as the range. As shown in Fig. 3(a), $Tr_1$ does not have any point neighbor. So both $Tr_1.NT$ and $Tr_1.tlist$ are null. $Tr_5$ instead acquires one point neighbor $p_2^1$ at timebin 1. So the record $< Tr_2, [t_1] >$ is created for $p_2^1$ and inserted into $Tr_5.NT$ as shown in Fig. 10(a). Since only one point neighbor is acquired (<*k*), timebin $t_1$ is not a neighboring timebin of $Tr_5$. $Tr_5.tlist$ thus is null. Similarly INC proceeds to detect point neighbors using a range query and updates the data structures at each time bin. After timebin $t_4$ is processed, the final neighboring information for $W_1$ is established as shown in Fig. 4(c).

**Trajectory Outlier Detection.** Then using $Tr_i.tlist$ and $Tr_i.NT$ INC can quickly detect both classes of trajectory outliers. As shown in Fig. 3(a), $Tr_1$ and $Tr_5$ can be immediately identified as *PN-Outliers*, because the cardinality of $Tr_1.tlist$ and $Tr_5.tlist$ (♯ of neighboring timebins) is smaller than $thr_t$ (3) respectively.

Similarly the *TN-Outliers* can be detected based on the neighbor table $Tr_i.NT$. $Tr_1$ is a *TN-Outlier* because it does not have any trajectory neighbor in $Tr_1.NT$. For that we only need to examine $Tr_i.NT$ to find the trajectories having no less than $thr_t$ point neighbors with $Tr_i$. $Tr_5$ however is not a *TN-Outlier*, because it has two trajectory neighbors $\{Tr_2, Tr_6\}$ ($k$ = 2) according to $Tr_5.NT$.

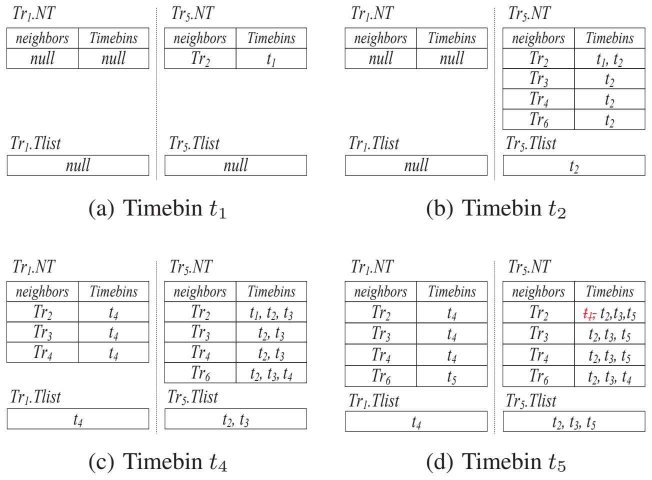


(a) Window $W1$ (b) Window $W2$

**Figure 3: An example of two consecutive windows**

**Data Structure Update.** After the new timebin $t_5$ arrives, the window slides from $W1$ to $W2$ (Fig. 3(b)). Since timebin $t_1$ has expired, all information of $t_1$ will be removed from $Tr_i.NT$ and $Tr_i.tlist$. The INC algorithm only needs to detect the point neighbors for each trajectory $Tr_i$ at the new timebin $t_5$. This again is done by a range query search. The corresponding data structures are then updated (see Fig. 4(d)).

**Outlier re-examination.** Then the status of all the trajectories will be re-examined by again checking the $Tr_i.tlist$ and $Tr_i.NT$ structures. In the new window $W_2$, $Tr_1$ is still a *PN-Outlier*. However $Tr_5$ has evolved into a *PN-Inlier*, because $Tr_5$ acquires an new neighboring timebin at $t_5$. Thus now $|Tr_5.tlist| = thr_t$ (3).



(a) Timebin $t_1$

$Tr_1.NT$

| neighbors | Timebins |
|---|---|
| null | null |

$Tr_1.Tlist$: null

$Tr_5.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | $t_1$ |

$Tr_5.Tlist$: null

(b) Timebin $t_2$

$Tr_1.NT$

| neighbors | Timebins |
|---|---|
| null | null |

$Tr_1.Tlist$: null

$Tr_5.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | $t_1, t_2$ |
| $Tr_3$ | $t_2$ |
| $Tr_4$ | $t_2$ |
| $Tr_6$ | $t_2$ |

$Tr_5.Tlist$: $t_2$

(c) Timebin $t_4$

$Tr_1.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | $t_4$ |
| $Tr_3$ | $t_4$ |
| $Tr_4$ | $t_4$ |

$Tr_1.Tlist$: $t_4$

$Tr_5.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | $t_1, t_2, t_3$ |
| $Tr_3$ | $t_2, t_3$ |
| $Tr_4$ | $t_2, t_3$ |
| $Tr_6$ | $t_2, t_3, t_4$ |

$Tr_5.Tlist$: $t_2, t_3$

(d) Timebin $t_5$

$Tr_1.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | $t_4$ |
| $Tr_3$ | $t_4$ |
| $Tr_4$ | $t_4$ |
| $Tr_6$ | $t_5$ |

$Tr_1.Tlist$: $t_4$

$Tr_5.NT$

| neighbors | Timebins |
|---|---|
| $Tr_2$ | ~~$t_1$,~~ $t_2, t_3, t_5$ |
| $Tr_3$ | $t_2, t_3, t_5$ |
| $Tr_4$ | $t_2, t_3, t_5$ |
| $Tr_6$ | $t_2, t_3, t_4$ |

$Tr_5.Tlist$: $t_2, t_3, t_5$

**Figure 4: Data structures of INC algorithm**

$Tr_1$ is still reported as a *TN-Outlier* in $W_2$, because it still has no trajectory neighbor. $Tr_5$ remains a *TN-Inlier*, since it has four trajectory neighbors as shown in $Tr_5.NT$ of Fig. 4(d).

**Complexity Analysis.** The INC algorithm detects both types of outliers by first running a range query search for each trajectory at each new timebin. The complexity is $O(n^2)$. Then INC detects *PN-Outlier* by checking the cardinality of each $Tr_i.tlist$. The cost is $O(n)$. So the overall complexity of *PN-Outlier* detection is dominated by the range query whose complexity is $O(n^2)$.

*TN-Outlier* detection is much more expensive than *PN-Outlier* due to the extra cost of having to traverse the neighbor table $Tr_i.NT$ to discover trajectory neighbors. Its worst case complexity is $O(n^2)$. Therefore the overall complexity of *TN-Outlier* detection is determined by both the range query search and the lookup of the full neighbor relationship produced by range query.

## 4. OPTIMIZED DETECTION FRAMEWORK

Although the INC algorithm for the current window detection fully reuses the neighbor relationships collected in the previous window, it still incurs high computational costs when the number (*n*) of the trajectories is large. This performance bottleneck is driven due to the $O(n^2)$ complexity of the expensive neighbor range query search and the corresponding neighbor lookup operation as shown in the complexity analysis above. To further drive down the CPU and memory costs, we now present our *minimal examination* ($MEX$) optimized framework. By proposing three innovative optimization principles, namely *minimal support examination*, *time-aware examination*, and *lifetime-triggered detection*, the MEX framework thoroughly eliminates the performance bottleneck of the INC algorithm caused by its "range query search first, outlier examination next" strategy.

### 4.1 Key Observations

Each trajectory in a window *W* is eventually reported as either *outlier* or *inlier*. A trajectory will be labeled as inlier if sufficient neighbor evidence has been acquired for this object. This fact leads to an important observation. That is, to identify whether a trajectory is a neighbor-based inlier, we may not need to find out all its neighbor information. Instead a potentially small subset of the full *neighbor evidence* often can be sufficient to prove that it is an *inlier*. Similarly a small subset of the *non-neighbor evidence* might also in some cases found to be sufficient to classify a trajectory as an *outlier*. To characterize the least amount of information needed to prove $Tr_i$'s status we define the concept of *Minimal Support*.

**Definition** 5. ***(Minimal Support).*** *Given a stream trajectory outlier detection query $Q$ and a trajectory set $DB_{Tr}$ in a window $W_c$, for a trajectory $Tr_i \in DB_{Tr}$, if the evidence pair $(TR, T)$ composed of trajectory points and timebins with $(TR = \{Tr_1, \ldots Tr_x, \ldots Tr_m \,|Tr_x \in DB_{Tr}(1 \le x \le m)\},\ T = \{t_1, \ldots t_j \ldots t_n \,|t_j \in [W_c.T_{Start}, W_c.T_{End}](1 \le j \le n)\})$ is sufficient to validate that $Tr_i$ is either inlier or outlier, and for any subset $TR' \subseteq TR$ and $T' \subseteq T$, the pair $(TR', T')$ is not sufficient to prove $Tr_i$'s status, then $(TR,\ T)$ is a minimal support of $Tr_i$ in $W_c$.*

The above structure of *minimal support* provides the minimal amount of evidence for identifying both *inliers* and *outliers*. This minimal support concept guides us to propose the *minimal support examination* principle (Sec. 4.2.1) to optimize the trajectory outlier detection process, in particular to reduce the neighbor search and lookup costs related to the range queries.

We also observe that the *minimal support* is not unique for a trajectory in each window. That is, several distinct minimal support sets may exist because the definition of the outlier only imposes a constraint on the neighbor evidence count, but not on which particular neighbor evidence must be utilized.

Next we introduce the second optimization principle, namely *Predicted Support* in Lemma 2. This principle guide our MEX framework to discover the *best minimal support* for each trajectory.

**Lemma** 2. ***Predicted Support:*** *Given a stream trajectory outlier detection query Q, if the evidence pair $(TR, T)$ is a minimal support of trajectory $Tr_i$ in $W_c$ as per Def. 5, then $(TR, T)$ is also a minimal support of $Tr_i$ in the subsequent windows from $W_{c+1}$ to $W_{c+x}$, where $W_{c+x}.T_{start}$=$Min(T)$ and $Min(T)$ represents the minimal timebin in $T$.*

PROOF. By Defs. 3 and 4, the criteria we can use to determine the status of a trajectory $Tr_i$ remains constant in each window. Therefore as the stream slides to a new window $W_{c+1}$, if no element of a minimal support set *MS* of $Tr_i$ expires, *MS* is still sufficient to determine the status of $Tr_i$ in $W_{c+1}$. Since $W_{c+x}$ is the last window in which all elements of *MS* are guaranteed to survive, $(TR, T)$ is still a minimal support for $Tr_i$ in the windows from $W_{c+1}$ to $W_{c+x}$. $\square$

Lemma 2 reveals two promising opportunities for optimizing stream trajectory outlier detection.

First, the status of $Tr_i$ can be predicted in certain future windows without first having to observe all data points of these windows. This insight inspires us to introduce the *lifetime-trigged detection optimization* principle in Sec. 4.2.3.

Second the more windows a minimal support of $Tr_i$ covers, the less re-evaluation effort will be needed for $Tr_i$. Therefore acquiring the minimal support covering the longest window sequence with minimal CPU costs is critical for stream trajectory outlier detection. This observation guides us to propose the *time-aware examination optimization* as highlighted in Sec. 4.2.2.

### 4.2 Optimization Principles

Based on the above *minimal support* and *predicted support* observations, we now are ready to propose three fundamental principles for optimizing the stream trajectory outlier detection process.

#### 4.2.1 *Minimal Support Examination Optimization*

Leveraging the minimal support observation (Def. 5) the *minimal support examination* (MSE) principle eliminates the complete and thus expensive point neighbor search for each trajectory adopted by the INC algorithm.

PRINCIPLE 1. *Given a trajectory outlier detection query $Q$ and the trajectory set $DB_{Tr}$ of window $W_c$, when evaluating a trajectory $Tr_i \in DB_{Tr}$, the **minimal support examination** principle suggests that the status determination process of $Tr_i$ can be terminated as soon as k neighbors have been found.*

This principle aims to prove the status of a given trajectory $Tr_i$ by only discovering a small subset of its neighbors instead of searching through its complete neighborhood to classify all points. As shown in Corollaries 1 and 2, this principle is equally applicable to both *PN-Outlier* and *TN-Outlier* detection, although they each have different concepts of neighbor respectively.

**Corollary** 1. *Given a PN-Outlier detection query $Q$ in window $W_c$, a timebin $t_j$ can be safely classified as a **neighboring timebin** of $Tr_i$ if k point neighbors have been acquired for $Tr_i$ at $t_j$.*

**Corollary** 2. *Given a TN-Outlier detection query $Q$ in window $W_c$, a trajectory $Tr_i$ can be safely classified as a **TN-Inlier** if $k$ trajectory neighbors have been identified for $Tr_i$.*

The proof of Corollaries 1 and 2 directly follows the definition of neighboring timebin (Def. 3) and $TN\text{-}Inlier$ (Def. 4) respectively.

The MSE principle enables us to design a lightweight neighbor search operation called *Examining* to replace range query.

**Definition** 6. *Given a trajectory $Tr_i$ in the window $W_c$, examining is an operation that evaluates the distance between the trajectory points of $Tr_i$ and the corresponding points of other trajectories until either k neighbors (either point or trajectory neighbors) are acquired or $Tr_i$'s entire neighborhood has been evaluated.*

Since the neighbor-count threshold *k* is much smaller than the average number of the neighbors we expect each trajectory may have, this examining operation is fundamentally more efficient than the full range query search.

### 4.2.2 Time-aware Examination Optimization

Our second optimization principle, called the *time-aware examination* (TAE) further optimizes the process of acquiring the timebin set *T* of the minimal support pair $(TR, T)$.

PRINCIPLE 2. ***Time-aware Examination:*** *Given the detection query Q and a trajectory $Tr_i$ in the current window $W_c$, the examining operation should identify the neighbor evidence for trajectory $Tr_i$ from the most recent to the earlier un-evaluated timebins until either neighbor evidence is found at $thr_t$ timebins or non-neighbor evidence are identified at $(w - thr_t + 1)$ timebins.*

The TAE principle has two implications. First, the examining operation should evaluate the un-evaluated timebins in the **latest time first order**. Second, TAE provides the criteria for the examining operation to *terminate the neighboring timebin search process*. Similar to the MSE principle, TAE can equally be applied to $PN\text{-}Outlier$ and $TN\text{-}Outlier$.

For a $PN\text{-}Outlier$ detection query w.r.t parameters $(d, k, thr_t)$, the status of a trajectory $Tr_i$ can be determined once either of the termination conditions shown in the following lemmas is reached.

**Lemma** 3. *Given a $PN\text{-}Outlier$ detection query Q, if a trajectory $Tr_i$ has already acquired t neighboring timebins in the current window $W_c$, then $Tr_i$ is an $PN\text{-}Inlier$ in $W_c$ as well as in the subsequent ($t_l$-$W_c.T_{Start}$) windows, where $t_l$ is the oldest neighboring timebin acquired by $Tr_i$.*

PROOF. First, by Def. 3, $Tr_i$ is a $PN\text{-}Inlier$ in $W_c$. Second, $\forall$ timebin $t_j \in T$, $t_j \geq t_l$. Therefore the minimal timebin in T (*Min(T)*) equals to $t_l$. By Lemma 2, $Tr_i$ thus is also a $PN\text{-}Inlier$ in the subsequent windows from $W_{c+1}$ to $W_{c+x}$, where $W_{c+x}.T_{Start} = t_l$. Therefore, $Tr_i$ is guaranteed to be a $PN\text{-}Inlier$ in the next ($t_l$-$W_c.T_{Start}$) windows. □

However, the status of $Tr_i$ after window $W_{c+x}$ (after $t_l$ expires) is uncertain because the remaining evidence ($k$-1 neighboring timebins) is no longer sufficient to prove its status. Therefore, $t_l$ is the furthest foreseeable timebin until which the status of $Tr_i$ is guaranteed to be certain. We call $t_l$ the **closed time** of $Tr_i$ to be a *PN-Inlier*.

Similarly the TAE principle is effective on the $TN\text{-}Outlier$ (Lemma 4).

**Lemma** 4. *Given a $TN\text{-}Outlier$ detection query Q, if $Tr_i$ has already acquired $thr_t$ neighboring timebins with a given trajectory $Tr_m$ in the current window $W_c$, then $Tr_i$ and $Tr_m$ are guaranteed to be trajectory neighbors in $W_c$ and in the subsequent ($t_l$-$W_c.T_{start}$) windows, where $t_l$ is their oldest neighboring timebins.*

For Lemma 4, timebin $t_l$ is called the **closed time** of $Tr_i$ to be trajectory neighbor of $Tr_m$. Due to space restriction, the proof is omitted here.

In summary, TAE aims to not only find the minimal timebin set sufficient to determine the trajectory's status, but also guarantees that the identified evidence is reused for the longest subsequent number of windows. Together the TAE and MSE principles guide the *examining operation* to effectively gather the optimal *minimal support* for each trajectory rather than conducting expensive and wasteful range query searches.

### 4.2.3 Lifetime-triggered Detection Optimization

The above two principles focus on how to optimize each single examining operation. To further reduce the computational costs, we now introduce another optimization principle regarding to the minimization of the examining frequency, termed *lifetime-triggered detection optimization* (LTD).

First, we define the *lifetime* concept of a trajectory.

**Definition** 7. ***(Lifetime).*** *Given a trajectory $Tr_i$ in the current window $W_c$, if $Tr_i$ is guaranteed to keep its status (being inlier or outlier) until timebin $t_{life}$ ($W_c.T_{Start} \leq t_{life} \leq W_c.T_{End}$) expires according to the identified evidence, then $t_{life}$ is called the **lifetime** of $Tr_i$ in the current window $W_c$.*

In other words the lifetime of $Tr_i$ indicates the duration of its current status (outlier or inlier).

Next we introduce a methodology for identifying the lifetime of a given trajectory for each of the two neighbor-based trajectory outlier definitions respectively.

**Lemma** 5. *Given a trajectory $Tr_i$, the lifetime of $Tr_i$ to be a $PN\text{-}inlier$ is the closed time of $Tr_i$ in $W_c$.*

PROOF. Lemma 5 can be easily proven by Lemmas 3. If $Tr_i$ is a $PN\text{-}inlier$, $Tr_i$ will remain to be inlier until its close time expires by Lemma 3. □

**Lemma** 6. *Given a trajectory $Tr_i$, the lifetime of $Tr_i$ to be a $TN\text{-}inlier$ is $min\{Tr_j.closedTime | Tr_j \in TN\}$, where TN is the trajectory neighbor set of $Tr_i$ in $W_c$.*

PROOF. Omitted due to space constrain. □

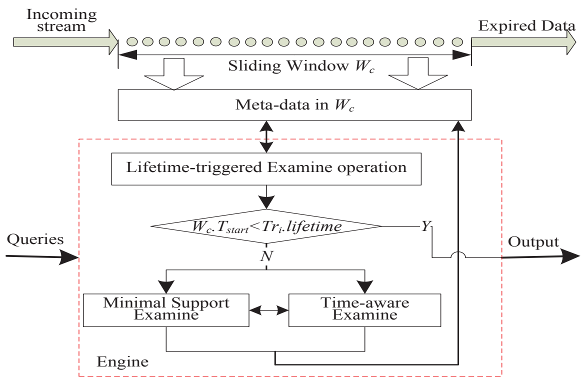


**Figure 5: Architecture of MEX framework**

Next we define the LTD optimization principle based on the above lifetime concept.

PRINCIPLE 3. ***Lifetime-triggered Detection:*** *Given a trajectory $Tr_i$, the examining operation will be triggered on $Tr_i$ if and only if the lifetime of $Tr_i$ holds with: $Tr_i.life \leq W_c.T_{start}$.*

By the LTD principle, the status of a trajectory is re-examined only when its lifetime expires. This effectively transforms the **continuous** query execution into **lifetime-triggered** execution. Whenever a trajectory is being re-examined, the examining operation which incorporates both MSE and TAE optimizations can be exploited to re-establish the minimal support in the new window. It does so by only acquiring enough new evidence rather than building a new minimal support from scratch. We call this enhanced examining operation *lifetime-aware examining* or in short LIFT.

## 4.3 Minimal Examination Framework

The high-level architecture of our MEX framework is depicted in Fig. 5. Similar to the INC algorithm, the MEX framework utilize the $Tr_i.tlist$ and $Tr_i.NT$ structures to store the meta-data. Therefore it also leverages the overlap of the adjacent windows in the sliding window stream.

Then the MEX framework continuously detects the trajectory outliers by conducting the LIFT operation on each trajectory. Given a trajectory $Tr_i$, the LIFT operation is not triggered if $W_c.T_{Start} < Tr_i.lifetime$. Once triggered, then LIFT employs both of the MSE and TAE principles to establish the new minimal support. The status and the lifetime of $Tr_i$ are also updated based on the new minimal support. Finally it outputs the outliers of the current window. The detailed description of the algorithm specific to each of the trajectory outlier definitions is described in the following section.

## 5. MEX-BASED TRAJECTORY OUTLIER DETECTION ALGORITHMS

### 5.1 The Optimized PN-Outlier Algorithm

Alg. 1 shows the optimized *PN-Outlier* detection algorithm as extension of the MEX framework, named *PN-Opt*. For each trajectory $Tr_i$, $Tr_i.untlist$ maintains the unchecked timebins in the current window, while $Tr_i.ntlist$ and $Tr_i.tlist$ correspond to the lists of non-neighboring and neighboring timebins of $Tr_i$ respectively. The lifetime-triggered detection optimization is employed first (line 1). That is, only the trajectories whose lifetime has expired are re-examined in the current window. For these triggered

**Algorithm 1** PN-Opt algorithm(*PN-Outlier* detection using MEX framework)

```
Input: Trajectory Set $DB_{Tr}$, the current window $W_c$, parameters: $d$, $k$ and $t$.
Output: outliers
 1: Get $DB_{lt} \leftarrow (W_c.T_{Start} - 1).triggered$;
 2: for each $Tr_i \in DB_{lt}$ do
 3:     for each $t_j \in Tr_i.untlist$ from $W_c.T_{End}$ to head do
 4:         if (true == $Tr_i.LIFT.MSEInHistory(t_j)$) then
 5:             $Tr_i.ntlist.add(t_j)$;
 6:         end if
 7:         if (true == $Tr_i.LIFT.MSEInNewTrjaectory(t_j)$) then
 8:             $Tr_i.tlist.add(t_j)$;
 9:         else
10:             $Tr_i.ntlist.add(t_j)$;
11:         end if
12:         $Tr_i.LIFT.TAE()$
13:     end for
14:     $Tr_i.updateLifetime()$;
15:     $Lifetime.triggered.add(Tr_i)$;
16: end for
```

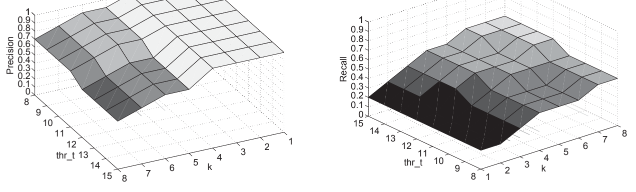
trajectories that must be examined, *PN-Opt* utilizes the MSE optimization principle to acquire new neighboring timbins (line 4). More specifically *PN-Opt* first checks whether $Tr_i$ can acquire $k$ point neighbors in the new timebin by searching through the trajectories who have been neighbors with $Tr_i$ in at least one other timebin. The other remaining trajectory points will be tested only if $Tr_i$ still has not acquired $k$ point neighbors (line 7). Then the TAE principle will be utilized to determine whether the neighboring timebin search process should be terminated, namely whether $thr_t$ timebins have been acquired in the new window (lines 3, 12). Finally, the lifetime of $Tr_i$, utilized by the LTD principle to determine whether the status of $Tr_i$ should be re-examined, is updated (lines 14,15).

### 5.2 The Optimized TN-Outlier Algorithm

**Algorithm 2** TN-Opt algorithm (*TN-Outlier* detection using MEX framework)

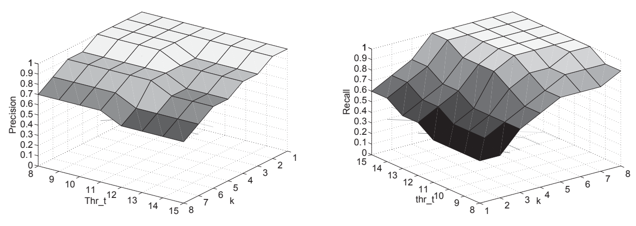
```
Input: Trajectory Set $DB_{Tr}$, the current window $W_c$, parameters: $d$, $k$ and $t$.
Output: Outliers
 1: Get $DB_{lt} \leftarrow (W_c.T_{Start} - 1).triggered$;
 2: for each $Tr_i \in DB_{lt}$ do
 3:     for each $Tr_m \in Tr.NT.keys$ do
 4:         $Tr_i.LIFT.TAE(Tr_m)$;
 5:         $Tr_i.LIFT.MSE()$;
 6:     end for
 7:     if (false == $Tr_i.getkNeighbors()$) then
 8:         for each $Tr_m \in (DB_{Tr} - Tr.NT.keys)$ do
 9:             $Tr_i.LIFT.TAE(Tr_m)$;
10:             $Tr_i.LIFT.MSE()$;
11:         end for
12:         if (false == $Tr_i.getkNeighbors()$) then
13:             $Tr_i.status$='outlier';
14:         end if
15:     end if
16:     $Tr_i.updateLifetime()$;
17:     $Lifetime.triggered.add(Tr_i)$;
18: end for
```

Alg. 2 utilizes the MEX framework to solve the *TN-Outlier* detection problem. Here, $Tr_i.NT$ denotes the neighbor table of $Tr_i$. Each element of $Tr_i.NT$ contains three parts: $tlist$, $ntlist$, $untlist$. $untlist$ maintains the unchecked timebins between $Tr_i$ and some trajectory $Tr_j$, while $ntlist$ and $tlist$ are the lists of non-neighboring and neighboring timebins between $Tr_i$ and $Tr_j$. Similar to Alg. 1, *TN-Opt* algorithm also employs the LTD first to check if the lifetime of $Tr_i$ has expired (line 1). If so, then *TN-Opt* assesses the status of $Tr_i$ again.



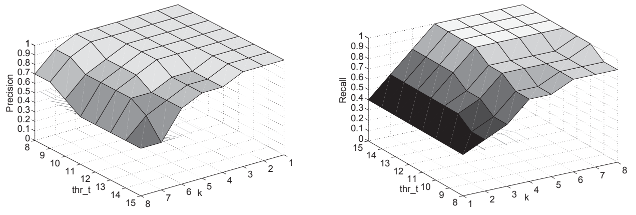
(a) Precision (b) Recall

**Figure 6: PN-Outlier (Taxi data)**



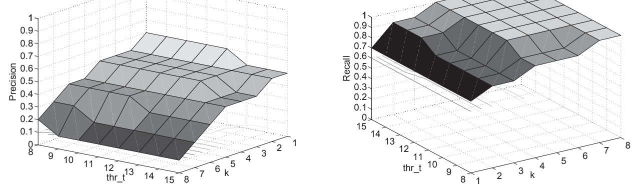
(a) Precision (b) Recall

**Figure 7: TN-Outlier (GMTI data)**



(a) Precision (b) Recall

**Figure 8: PN-Outlier (Taxi data)**



(a) Precision (b) Recall

**Figure 9: TN-Outlier (Taxi data)**

*TN-Opt* first applies the TAE optimization principle (lines 4,9) to test whether a given trajectory $Tr_j$ is a trajectory neighbor of $Tr_i$. Starting from the latest timebin, *TN-Opt* keeps testing the trajectory points at the unchecked timebins in *untlist* between $Tr_i$ and $Tr_j$ until point neighbors at *t* timebins are found. By the MSE principle (lines 5, lines 10), the trajectory neighbor search process stops immediately after $Tr_i$ acquires sufficient (*k*) trajectory neighbors or all the trajectories have been tested. Finally, the lifetime of $Tr_i$ is updated (lines 16,17).

## 6. EXPERIMENTAL EVALUATION

### 6.1 Experimental Setup

All algorithms are implemented in JAVA on CHAOS stream engine [6]. Our experiments are performed on a PC with 3.4G Hz Intel i7 processor and 6GB memory, which runs Windows 7 OS.

**Real Datasets.** We used two real streaming datasets. The *Taxi Dataset* is the real GPS trajectory data generated by 10,357 taxis in a period from Feb. 2 to Feb. 8, 2008 in Beijing [16] [17]. The total number of points in this dataset is about 15 million. The average time interval between two points is around 177 seconds. To model timebins, we interpolate the time granularity to 1 minute per timebin. The **GMTI** (Ground Moving Target Indicator) dataset [3] records the real-time trajectories of 150 moving objects gathered by 24 different data ground stations or aircraft in 6 hours. It has around 100,000 records regarding the information of vehicles and helicopters moving in a certain geographic region. In our experiment, we used all 14 dimensions of GMTI while detecting outliers based on targets' latitude and longitude.

**Metrics & Measurements.** We evaluate both the effectiveness of our outlier definitions and the efficiency of our MEX outlier detection algorithms.

For the effectiveness evaluation, we measure the quality of reported outliers by *Precision* and *Recall* as follows:

$Precision = \frac{|R_o \cap D_o|}{|D_o|}$, $Recall = \frac{|R_o \cap D_o|}{|R_o|}$

where $R_o$ denotes the set of annotated outliers in a dataset, namely the real outliers and $D_o$ the outliers detected by our algorithms.

For the efficiency evaluation, we measure two metrics common for stream systems, namely CPU resources and memory consumption. Experiments are conducted on 1,000 windows for the Taxi dataset. Both metrics are averaged over all windows.

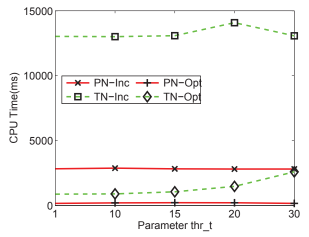
### 6.2 Effectiveness Evaluation

We evaluate the effectiveness of the new proposed outlier definitions by measuring the *precision* and *recall* on the real GMTI and taxi datasets. For the GMTI dataset, we use the outlier set manually labeled by the experts familiar with the data as the ground truth $R_o$. For the Taxi dataset the ground truth outlier set $R_o$ is instead produced by a user study. In this user study 100 sets of trajectories were selected from the taxi dataset. Each set contains 10 trajectories with 30 consecutive 6-minutes window. We invited 50 users from both WPI and USTB. The users were divided into 5 groups, with 10 trajectory sets assigned to each group. The users were asked to mark trajectories in each set that they believe are most likely to be outliers. Each trajectory marked by at least 5 users is labeled as a "real" outlier.

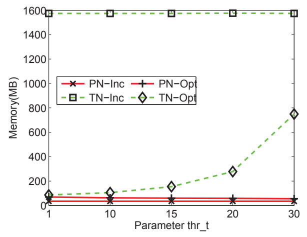
We vary the thresholds *k* and $thr_t$ to investigate how *Precision* and *Recall* are impacted. The *d* threshold is fixed as 200 meter for GMTI data and 300 meter for Taxi data.

**GMTI data**. The results for GMTI data are shown in Figs. 6 and 7. From Figs. 6(a) and 7(a) the *Precision* of both *PN-Outlier* and *TN-Outlier* is nearly 100% once the parameters *k* and $thr_t$ fall in a rather large range. Fig. 7(b) shows that the *Recall* of *TN-Outlier* is also good and robust. However the *Recall* of *PN-Outlier* (Fig. 6(b)) is much worse than *TN-Outlier*. In the GMTI dataset the moving objects (soldiers) are divided into 3 military units. The members of each unit are expected to operate in a team. If they do not continuously operate together, they will be labeled as outliers even if they happen to stay close to others at times. *TN-Outlier* perfectly covers this scenario. However sometimes *PN-Outlier* might fail to locate the soldier separated from his own unit, since *PN-Outlier* would not classify an object as outlier if there are a sufficient number of moving objects in its vicinity.

**Taxi data**. We also investigate the effectiveness of *PN-Outlier* and *TN-Outlier* for the Taxi data. As shown in Figs. 8(a) and 8(b) *PN-Outlier* shows nearly 100% *Precision* and *Recall* when the parameters *k* and $thr_t$ varies in a large range. However the *Precision* of *TN-Outlier* is worse than *PN-Outlier* (Fig. 9(a)), although its *Recall* is as good as *PN-Outlier* (Fig. 9(b)). In the user study the users tend to classify a taxi as an outlier if it always moves alone. Intuitively taxis do not necessarily move together with others as a group. The behavior of a taxi driver will be considered as abnormal only if he always operates in areas that other drivers rarely visit. This scenario fits *PN-Outlier* better than *TN-Outlier*, since

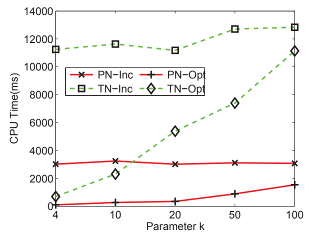


(a) CPU time

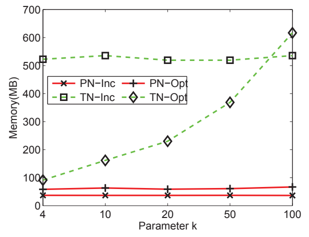


(b) Memory

**Figure 10: Varying $thr_t$ on taxi data**



(a) CPU time



(b) Memory

**Figure 11: Varying $k$ on taxi data**

*TN-Outlier* tends to misclassify the taxis lack of consistent companions as outliers.

In summary the above empirical study confirms the effectiveness of our new proposed *PN-Outlier* and *TN-Outlier* definitions in capturing the moving object outliers with only a very loose parameter setting requirement. Furthermore it shows that *PN-Outlier* and *TN-Outlier* best cover distinct application categories with different similarity requirement to be neighbors.

## 6.3 Efficiency Evaluation

Next we evaluate the efficiency of our outlier detection algorithms using the Taxi data. The performance of MEX algorithms is compared with the INC algorithms. We denote the INC solution for $PN\text{-}Outlier$, $TN\text{-}Outlier$ detection as *PN-INC*, *TN-INC* and the *MEX*-based solution as *PN-Opt* and *TN-Opt* respectively. We vary the most important parameters, to (1) assess the impact of our *MEX* framework versus the INC baseline, (2) evaluate sensitivity of parameter variations on each method.

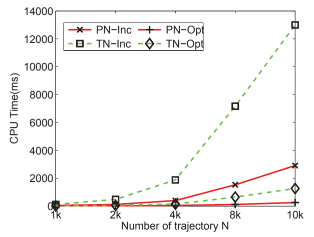
### 6.3.1 Varying Timebin Count Threshold $thr_t$

**CPU Resources.** First, we evaluate the effect of varying the timebin count threshold $thr_t$ from 1 to the full window size. This varies the definition from very relaxed (that is, one trajectory $Tr_i$ is inlier for instance if only in 1 of $w$ timebins $tr_i$ has the needed neighbors) to very strict ($Tr_i$ must have $k$ neighbors in all w timebins to be considered an inlier). We fix the window size to 30, $k$ to 4, and $d$ to 200 meters. As shown in Figs. 10, our MEX-based algorithms are superior to the corresponding basic INC-based solutions w.r.t the CPU time in all cases. In particular, *PN-Opt* is 117 times faster than *PN-INC* (Fig. 10(a)). When $thr_t$ is equal to the full window, the outliers rates of the two definitions are at their highest, namely 5% and 20% respectively. However, even in these cases the optimized algorithms still outperform their corresponding counterparts 31 and 5 fold.

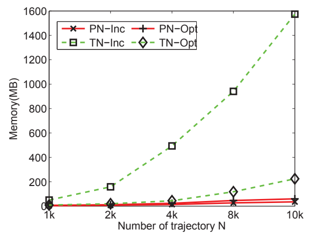
We also notice that INC algorithms are not very sensitive to $thr_t$, since INC always tests all timebins of each trajectory. Whereas by employing the TAE optimization principle (Sec. 4.2.2) MEX significantly reduces the distance calculation by only acquiring the minimal yet sufficient neighboring timebins.

**Memory Resources.** We see similar positive trends also for the memory usage. That is, *TN-Opt* uses on average 21% of the memory consumed by its counterpart *TN-INC* algorithm. This can be explained by the fact that the MEX framework only maintains $k$ trajectory neighbors for $TN\text{-}Outlier$, while the INC solution aggressively stores all neighbor information. As expected, the memory consumption of *TN-Opt* increases as $thr_t$ increases, because more neighboring timebins have to be maintained for each trajectory neighbor as the required neighbor count $thr_t$ rises.

As shown in Fig. 10(b), the memory usage of both *PN-Opt* and *PN-INC* is very small. *PN-Opt* uses a little more memory than *PN-INC*. For each trajectory $Tr_i$, *PN-Opt* stores the trajectories which have point neighbors with $Tr_i$ in the history to reduce the trajectory search scope for acquiring new neighboring timebins. However this



(a) CPU time



(b) Memory

**Figure 12: Varying number $n$ of trajectories on taxi data**

extra memory leads to huge gains in CPU processing resources (at least 31 times faster than *PN-INC*).

### 6.3.2 Varying Neighbor Count Threshold $k$

Next we evaluate the performance of the four algorithms by varying the neighbor count threshold $k$ from 4 to 100. To be classified as an inlier the number of neighbors that a trajectory $Tr_i$ needs to discover increases from very few to a large number. That is, the inlier criteria for any trajectory $Tr_i$ changes from very relaxed to strict. We fix the window size to 30, $thr_t$ to 15, and $d$ to 200.

**CPU Resources.** Fig. 11(a) shows both MEX-based algorithms clearly outperform their counterparts. The two MEX-based algorithms save on average 91% and 56% of CPU time compared to the corresponding INC solution. As $k$ increases, the CPU time of the MEX-based solutions increases linearly since more neighbors have to be acquired to determine the status of a given trajectory. For the INC-based solutions instead we observe no sensitivity for varying $k$. This is because the INC-based solutions determine the status of $Tr_i$ always by first acquiring all neighbors with expensive complete range query searches no matter how large $k$ is. However, the MEX-based solutions still outperform INC-based algorithms even for the extreme case of $k = 100$. In this case the outlier rate is extremely high, namely 90%, and thus unrealistic.

**Memory Resources.** *TN-INC* only uses on average 55% of the memory compared to *TN-INC* (Fig. 11(b)). As $k$ increases, they need to store more trajectory neighbors. Thus their memory consumption increases. As $k$ increases to 100, *TN-Opt* uses more memory than *TN-INC*. Again this is an unrealistic setting we do not expect to see in practice, as surely 90% of all data would not be considered as "exceptions" (outliers) but rather as the norm.

### 6.3.3 Varying Number of Trajectories $n$

We evaluate the scalability of our algorithms in the number of trajectories they can simultaneously handle. In this experiment we randomly select from 1k up to 10k trajectories from the Taxi data. We fix the window size to 30, $k$ to 4, $thr_t$ to 15. To eliminate the effect of variations in the outlier rates, we stabilize the outlier rate in all cases to around 4% by slightly adjusting the distance threshold $d$ from 200 to 300 meters.

**CPU Resources.** As shown in Fig. 12(a), the MEX algorithms exhibit much better scalability than their INC-based counterparts.

As the number of trajectories increases, as expected all algorithms require more time to process more trajectories. However, the MEX algorithms outperform the INC-based ones more and more as the number of trajectories increases, because unlike the INC solution MEX avoids expensive range queries for the needed searches.

**CPU Resources.** As shown in Fig. 12(b), the memory consumption of all algorithms also increases as the number of trajectories increases, because more trajectory information must be stored. However, *TN-Opt* saves more memory than *TN-INC* as $n$ increases, while the memory usage of *PN-Opt* remains consistently small.

## 7. RELATED WORK

**Trajectory Outlier Detection in Static Datasets.** Knorr et al. [9] applied the distance-based outlier notion defined in [8] to spatio-temporal data. It first extracted features from the trajectories already located in the database and mapped the trajectories into a feature space. Then it relied on the distance in this feature space to determine the relationships among the trajectories. The status of each trajectory was evaluated and reported only once. This definition does not fit the requirement of streaming data. First the features of the stream trajectories keep evolving. Therefore no stable feature space exists. Furthermore the effect of the observed events keeps fading. Therefore the outliers have to be continuously reported in real time based on the latest events rather than reporting them only once based one the whole trajectory database.

Li et al. [12] proposed a classification-based trajectory outlier detection algorithm. In their algorithm, trajectories are represented using discrete pattern fragments called motifs. The set of motifs forms a feature space in which the trajectories are placed. Then a rule-based classifier is trained to classify the trajectories into either "normal" or "abnormal". This algorithm cannot be adapted to the streaming context since it requires an offline training stage and a labeled training dataset to train the classifier.

Lee et al. [11] proposed a two-step trajectory outlier detection approach. At the first step, each trajectory is partitioned into a sequence of t-partitions. Within this set of t-partitions, outlying t-partitions are determined based on distance or density based metrics. This work targets on a completely different problem, namely to discover unusual sub-trajectories within one single trajectory. Instead we aim to locate abnormal moving objects.

**Outlier detection over trajectory data streams.** Bu et al. [2] detected abnormal trajectory segments in the stream context with the assumption that a stream trajectory is locally continuous. That is, a trajectory is expected to behave consistently within a short time interval. They use a base window to constrain the part of a trajectory they are interested in into a trajectory segment. Then whether a given trajectory segment is said to be anomalous or not is based on the similarity to its own historical trajectory in a larger window. Thus this method, focusing on identifying an outlying trajectory segment of one particular moving object, cannot address our problem, namely identifying suspicious moving objects significantly deviating from other synchronously moving objects.

Liu et al. [14] studied causal interaction detection in traffic data streams. They first divide the city areas into regions. A graph is built by mapping each region to a vertex. The trajectories are then simplified into links connecting two corresponding regions. This much smaller dataset, i.e., a graph of regions, enables them to map their problem into frequent subgraph mining. That is, they determine the anomalous links in each time frame by comparing their load features of links in the graph (e.g., number of traversals) with those of their temporal neighbors. This method focuses on a totally different problem, namely discovering anomalous regions rather than abnormal moving objects.

**Trajectory clustering and pattern mining.** Other domains of mining moving object streams have been studied with clustering of trajectory the closest to outlier detection. Lee et al. [10] presented a clustering algorithm on discovering common sub-trajectories in static trajectory databases. Trajectories are first partitioned into a set of quasi-linear segments using the minimum description length principle. Then the line segments are grouped by a density-based clustering algorithm. Although this algorithm could potentially be applied in a reverse direction to detect *abnormal trajectory segment*, it cannot solve our problem. First, the trajectory segments falling in one cluster are not necessarily observed at the same time period, while in trajectory stream to online detect abnormal moving objects only the "concurrent" behavior of the objects should be considered. Second, our goal is to continuously detect abnormal *moving objects* at real time rather than locate the unusual *segments* of the trajectories known apriori.

## 8. CONCLUSIONS

In this work we focus on the detection of abnormal moving objects over massive-scale trajectory streams. After analyzing the requirements of stream trajectory applications, we propose novel neighbor-based trajectory outlier definitions. Our empirical study on the GMTI and Taxi data shows that our definitions can effectively capture moving object outliers in different scenarios. Furthermore we design an optimized MEX strategy scalable to big data trajectory streams to detect the new classes of outliers, rendering moving object outliers detection practical in real time applications.

## 9. REFERENCES

[1] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foudations and query execution. The VLDB Journal, 15(2):121–142, July 2006.

[2] Y. Bu, L. Chen, A. W.-C. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In Proceedings of SIGKDD 2009, pages 159–168. ACM, June 2009.

[3] J. N. Entzminger, C. A. Fowler, and W. J. Kenneally. Jointstars and gmti: Past, present and future. IEEE Trans. on Aerospace and Electronic Systems, 35(2):748–761, Apr 1999.

[4] D. Freedman, R. Pisani, and R. Purves. Statistics. W. W. Norton & Company, 2007.

[5] Y. Ge, H. Xiong, C. Liu, and Z.-H. Zhou. A taxi driving fraud detection system. In Proceedings of ICDM 2011, pages 181–190. IEEE, December 2011.

[6] C. Gupta, S. Wang, I. Ari, and M. Hao. Chaos: A data stream analysis architecture for enterprise applications. In Proceedings of CEC 2009, pages 33–40. IEEE, July 2009.

[7] D. M. Hawkins. Identification of Outliers. Springer, 1980.

[8] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In Proceedings of VLDB 1998, pages 392–403, August 1998.

[9] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. The VLDB Journal, 8(3-4):237–253, March 2000.

[10] J.-G. Lee and J. Han. Trajectory clustering: A partition and group framework. In Proceedings of SIGMOD 2007, pages 593–604. ACM, June 2007.

[11] J.-G. Lee, J. Han, and X. Li. Trajectory outlier detection: A partition-and-detect framework. In In Proceedings of ICDE 2008, pages 140–149. IEEE, April 2008.

[12] X. Li, J. Han, S. Kim, and H. Gonzalez. Roam: Rule- and motif-based anomaly detection in massive moving object data sets. In Proceedings of SDM 2007, pages 273–284. SIAM, April 2007.

[13] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. In Proceedings of VLDB 2010, pages 723–734, August 2010.

[14] W. Liu, Y. Zheng, and S. Chawla. Discovering spatio-temporal causal interactions in traffic data streams. In Proceedings of SIGKDD 2011, pages 1010–1018. ACM, July 2011.

[15] J. Yuan, Y. Zheng, X. Xie, and G. Sun. Driving with knowledge from the physical world. In KDD, pages 316–324, 2011.

[16] J. Yuan, Y. Zheng, X. Xie, and G. Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. Transactions on Knowledge and Data Engineering, 25(1):220–232, January 2013.

[17] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In GIS, pages 99–108, 2010.

[18] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. IEEE Data Engineering Bulletin, 33(2):32–40, June 2010.