Outlier Detection over Massive-Scale Trajectory Streams

YANWEI YU, Yantai University
LEI CAO, Massachusetts Institute of Technology
ELKE A. RUNDENSTEINER, Worcester Polytechnic Institute
QIN WANG, University of Science and Technology Beijing

10

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 outlier detection problem, especially along both the spatial and temporal dimensions, remains largely unexplored. In this work, we propose a rich taxonomy of novel classes of neighbor-based 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 two real-world datasets—the Beijing Taxi trajectory data and the Ground Moving Target Indicator data stream—and one generated Moving Objects dataset demonstrate the effectiveness of our taxonomy in effectively capturing different types of abnormal moving objects. Furthermore, we propose a general strategy for efficiently detecting these new outlier classes called the *minimal examination* (MEX) framework. The MEX framework features three core optimization principles, which leverage spatiotemporal as well as the predictability properties of the neighbor evidence to minimize the detection costs. Based on this foundation, we design algorithms that detect the outliers based on these classes of new outlier semantics that successfully leverage our optimization principles. Our comprehensive experimental study demonstrates that our proposed MEX strategy drives the detection costs 100-fold down into the practical realm for applications that analyze high-volume trajectory streams in near real time.

CCS Concepts: • **Information systems → Data stream mining;**

Additional Key Words and Phrases: Outlier detection, moving objects, trajectory stream

ACM Reference Format:
Yanwei Yu, Lei Cao, Elke A. Rundensteiner, and Qin Wang. 2017. Outlier detection over massive-scale trajectory streams. ACM Trans. Database Syst. 42, 2, Article 10 (April 2017), 33 pages.
DOI: http://dx.doi.org/10.1145/3013527

1. INTRODUCTION

Motivation. In recent years, location-acquisition devices such as GPS, smartphones, and RFID have become prevalent. These devices, which monitor the motion of vehicles, people, goods, services, and animals, are producing massive-volume high-speed trajectory streams. Many applications—from traffic management [Ge et al. 2011], security surveillance [Bu et al. 2009], scientific studies [Li et al. 2010], to mobile social networks

This work is supported in part by National Natural Science Foundation of China under Grant Nos. 61403328, 61502410, 61572419; the Key Research & Development Project of Shandong Province under Grant Nos. 2015GSF115009. This work is also supported by NSF grants IIS-1018443 and IIS-0917017.
Authors' addresses: Y. Yu, School of Computer and Control Engineering, Yantai University, Shandong, China, 264005; email: yuyanwei0530@gmail.com; L. Cao (corresponding author), CSAIL, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139; email: lcao@csail.mit.edu; E. A. Rundensteiner, Computer Science Department, Worcester Polytechnic Institute, Worcester, Massachusetts 01609; email: rundenst@cs.wpi.edu; Q. Wang, the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China 100083; email: wangqin@ies.ustb.edu.cn.

DOI: http://dx.doi.org/10.1145/3013527

[Zheng et al. 2010]—rely on continuously discovering abnormal objects in such trajectory streams to deliver critical decisions within an actionable time.

In security surveillance systems, visitors at a military base will be considered as outliers and thus a potential safety threat if they do not obey the strict order to stay together with their designated group members. In traffic management systems, a taxi driver will be classified as an outlier in terms of erratic behavior if the driver keeps changing lanes and by this continuously encounters new neighboring travel companions. Potentially this may help flag speeding, drunk driving, or other erratic behaviors of concern.

Similarly, if considering the price, volume, or gain of stocks at a particular time point as coordinates in a multidimensional space, then real-time stock quotes can be modeled as a trajectory stream. Analysts then may discover stocks with promising characteristics by detecting those outlier stocks whose performance trajectories dramatically deviate from those of other stocks in the same industry.

Challenges. In the applications just described, outliers can be characterized as moving objects that behave differently from the majority of the trajectories in the stream. Despite the importance of continuously detecting such types of outliers, to the best of our knowledge, this problem has been considered rarely in the previous literature.

In the streaming context, Bu et al. [2009] have defined criteria by which to label a trajectory *segment* of a single moving object as an outlier. Our work instead focuses on a more complicated problem: locating outlier objects in the trajectory stream populated with massive-scale moving objects. In Bu et al. [2009], a given trajectory is divided into equal-sized segments. A segment is said to be abnormal if it is not similar to the group of segments adjacent to it in time. This definition relies on *local continuity*. It assumes that each moving object tends to behave (relatively) consistent (stable) over time. However, in our context, whether one moving object is an outlier or not instead 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. Thus, they cannot be modeled by the local continuity property. Therefore, this approach cannot be applied to our problem.

In static spatiotemporal databases, commonly called time series data, a trajectory is said to be an outlier if it does not show the same global characteristics that the majority of the trajectories in the database feature [Knorr and Ng 1998; Lee et al. 2008]. Since all trajectories are known *a priori*, these techniques can rely on *expensive* offline preprocessing. Typically, they first mine all frequent patterns to build a model that captures the global characteristics of the dataset. In the second phase, this model is then used to classify each trajectory as being either an outlier or inlier. However, in our context—infinite continuous trajectory streams in which concept drift frequently arises—using one single (precomputed) model to continuously detect outliers would inevitably lead to inaccurate results. On the other hand, or worse yet, to continuously rebuilt such a 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 massive-scale trajectory streams, new semantics have to be defined to satisfy the requirements of streaming trajectory outlier detection. First, lightweight metrics are needed that are suitable for identifying outliers even in massive-scale trajectory streams, instead of complicated statistical models. These metrics must capture the key characteristics of moving objects. Second, given the dynamic nature of stream trajectories, these semantics 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 introduced, we still would require effective algorithms that, based on these detection semantics, can efficiently extract real-time insights from *high -volume* trajectory stream data, such

as streams of surveillance, stocks, or traffic data. Thus, strategies must be designed to efficiently discover these assets in the form of stream trajectory outliers online as stream data passes by.

Proposed Solution. In this article, 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 [Knorr and Ng 1998], which simply considers the physical distance between two objects at a given point in time, the trajectory neighbor concept captures the key properties of stream trajectories. *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 real-time trajectory applications reveals that applications vary in their particular synchronization requirements with respect to the neighbor relationships among multiple trajectories. We thus propose several variants of the trajectory neighbor concept with different synchronization regulations that empower users to customize the desired outlier semantics by controlling certain parameters. Different settings of these controlled parameters lead to three different classes of trajectory outlier semantics—in particular, *point neighbor–based outliers* (*PN-Outliers*), *trajectory neighbor–based outliers* (*TN-Outliers*), and *synchronized neighbor–based outliers* (*SN-Outliers*).

By measuring the *Precision* and *Recall* of proposed algorithms on several datasets from *GMTI* [Entzminger et al. 1999], *Taxi* [Yuan et al. 2013, 2010], to *MOD* [Brinkhoff 2002] data, our empirical study confirms that these newly introduced classes of semantics 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 all three 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 to confirm or reject the outlier status. 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 as soon as possible. Third, MEX establishes the *lifetime* concept signaling the farthest window up to which we can predict the status of a trajectory Tr_i based on the current trajectory neighbor evidence. This enables MEX to transform the periodical per window–based outlier detection process into a *lifetime-triggered* detection process, meaning that the detection process is run more rarely, only when absolutely necessary.

Our experimental study on the largest Taxi dataset shows that MEX can successfully handle up to one million moving objects per second on a standard desktop, rendering trajectory outlier detection practical even when applied to such massive-scale moving object streams.

Contributions. The main contributions of this work include the following: (1) We propose a taxonomy of neighbor-based trajectory outlier definitions that cover a wide spectrum of outlier semantics in the stream context. We then show that these semantics successfully model the abnormal behavior of moving objects in a rich diversity of stream trajectory applications. (2) Our empirical study using GMTI, Taxi, and MOD datasets demonstrates the robust *Precision* and *Recall* of our new proposed semantics. (3) We design a set of incremental algorithms to process all classes of outlier semantics in this proposed taxonomy. These algorithms effectively leverage the overlap of sliding windows. (4) We further enhance the incremental approach by proposing the

MEX framework equipped with three core optimization principles: minimal support examination, time-aware examination, and lifetime-triggered detection. (5) We design efficient algorithms for all three trajectory outlier definitions based on MEX, successfully driving the detection costs down by 100-fold.

Extension from the Conference Version. While this work is based on a conference article [Yu et al. 2014], the scope of the proposed work has been significantly extended.

—In the earlier conference article [Yu et al. 2014], two classes of trajectory outliers were proposed; in this article, we now propose a third class of distinct neighbor-based trajectory outlier semantics. The new class of semantics models a range of real-life trajectory stream applications that require stricter criteria in recognizing neighbors compared to the previous two classes (Section 3.2.3).
—We not only formally analyze the unique properties of the newly proposed trajectory definition but also reveal the intrinsic relationships among the different definitions. This analysis helps us to better understand the essence of the neighbor-based trajectory outlier taxonomy (Section 3.3).
—We extend our general trajectory outlier detection framework MEX to also cover this third definition of the neighbor-based trajectory outlier taxonomy in a unified manner. This further validates the generality of the MEX framework as a processing paradigm for trajectory outlier detection.
—We now give a formal complexity analysis of our proposed three optimization algorithms using the MEX framework (Section 6.4). In addition, we elaborate on complexity analysis of the INC algorithm. This is also validated by results of our extensive efficiency evaluation.
—We significantly extend the experimental evaluation, focusing on the *effectiveness* of all outlier definitions in our proposed taxonomy. First, in this article, we now evaluate the effectiveness of the newly proposed *SN-Outlier* definition on the two real datasets that had already been introduced in Yu et al. [2014]. On top of that, we adopt one additional large-scale dataset (the MOD dataset), with which all three definitions are also evaluated (Section 7.2).
—We conduct an additional experimental study to demonstrate the *efficiency* of the newly proposed *SN-Outlier* detection algorithm under the unified MEX framework (Section 7.3). Furthermore, we give a formal analysis of the performance of *SN-Outlier* detection.
—We conduct additional experiments to compare both the effectiveness of our three outlier definitions and the efficiency of our proposed detection algorithms against the state-of-the-art TRAOD [Lee et al. 2008]. TRAOD was proposed to detect outliers in static trajectory databases. In this revision, we extend it to support streaming trajectory data.

2. RELATED WORK

Trajectory Outlier Detection in Static Datasets. Knorr et al. [2000] applied the distance-based outlier notion defined in Knorr and Ng [1998] to spatiotemporal data. They first extract features from the trajectories that were already located in the database and then map these trajectories into a feature space. Then, they rely on the distance in this feature space to determine the relationships among the trajectories. The status of each trajectory is evaluated and reported only once. This approach does not fit our target of streaming data for several reasons. First, in the streaming context, the features of stream trajectories would continue to evolve. Therefore, no stable feature space exists. Furthermore, the effect of the previously observed events would fade over time. Therefore, outliers should be continuously reported in real time

based on the latest observed events rather than reporting them only once based on the final complete trajectory database.

Li et al. [2007] 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 requires an offline training stage as well as a labeled training dataset to train the classifier. Thus, it does not fit the dynamic streaming context that we target.

Lee et al. [2008] proposed a two-step trajectory outlier detection approach. In the first step, a trajectory is partitioned into a sequence of t-partitions. Within this set of t-partitions, t-partitions are determined to be outliers based on distance or density-based metrics. This work targets a problem different from ours, namely, to discover unusual subtrajectories within one single trajectory. Instead, we aim to locate abnormal moving objects within one stream of companion moving objects.

Zhang et al. [2011] proposed an isolation-based anomalous trajectory detection method to discover fraud driving patterns within a taxi's GPS traces. The method first groups all historical trajectories with a same source-destination cell-pair. They represent each trajectory as a sequence of cell symbols, then find the trajectories deviating from the same source-destination cell-pair frequent paths. With a similar objective, Ge et al. [2011] developed a taxi-driving fraud detection system based on historical trajectories to identify suspicious taxi driving. However, the system only detects anomalous trajectories after the trips are complete; thus, the data is effectively considered to be static.

Outlier Detection over Trajectory Data Streams. Bu et al. [2009] detect abnormal trajectory segments in a single trajectory produced by one particular continuously moving object. It makes the strong assumption that a stream trajectory is locally continuous. That is, a trajectory behaves consistently within any short time interval. They use a base window to partition a trajectory into trajectory segments. 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 single moving object, cannot address our problem: identifying suspicious moving objects significantly deviating from other synchronously moving objects within a similar space and at the same time.

Ge et al. [2010] propose a top-k evolving outlying trajectory detection method. They compute the outlier score in an accumulating fashion, based on the evolving direction and density of historical trajectories in which grid the trajectories passed.

Liu et al. [2011] studied causal interaction detection in traffic data streams. For that, they divide the city areas into regions. A graph is built by mapping each region to a vertex. The trajectories are then simplified by handling them at this coarse granularity, namely, mapped into links connecting two corresponding regions. This much smaller dataset, that is, a graph of regions, enables them to map their problem into a frequent subgraph mining problem. That is, they determine the anomalous links in each time frame by comparing the load features of links in the graph (e.g., the number of traversals) with those of their temporal neighbors. This method focuses on a problem totally different from ours, which is discovering anomalous historical links rather than abnormal moving objects.

Trajectory Clustering and Pattern Mining. Other classes of mining tasks on moving object streams have been studied, with clustering of trajectories being the closest to outlier detection. Lee and Han [2007] presented a *clustering* algorithm that discovers common subtrajectories in a static trajectory database. 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 segments*, it cannot solve our problem for several reasons. First, the trajectory segments that fall into the same cluster are not necessarily observed at the same time period, while in our trajectory stream context, we are interested in the "concurrent" and relative behavior of objects. Second, our goal is to continuously detect abnormal *moving objects* in near real-time rather than to locate the unusual *segments* of the trajectories prestored in the database.

Moreover, in recent years, mining trajectory patterns—such as Jeung et al. [2008], Li et al. [2010], Zheng et al. [2013, 2014], and Tang et al. [2012]—have been discussed to find groups of moving objects with similar motion patterns or behaviors, as described next.

A *Convoy pattern* in Jeung et al. [2008] is defined as a set of objects that move together (always falling in the same density-based cluster) during at least k contiguous time points. In Li et al. [2010], a *swarm pattern* is defined as a variation of the convoy pattern with one relaxed requirement, that is, those objects could form a swarm group even if they are not always moving together in a number of consecutive time points. In other words, moving objects could leave the swarm temporarily, and then come back into the swarm again.

Tang et al. [2012] also similarly propose the notion of traveling companion. They focus on the discovery of traveling companions in the context of streaming trajectories. Zheng et al. propose the *Gathering pattern* in Zheng et al. [2013, 2014] that captures congregations of moving individuals, which form a durable and stable area with high density. Similar to the swarm pattern, the members in the gathering pattern can enter and leave the group at any time. In general, their objective is to discover the clusters of the objects that move together in either static trajectory databases or dynamic trajectory streams. However, the topic of detecting moving object outliers is not discussed in any of these papers.

Although, theoretically, trajectories that do not belong to any cluster can be considered as outliers, utilizing such a clustering approach to detect outliers is neither effective nor efficient. First, outlier detection is not a counterpart of clustering [Chandola et al. 2009]. Even if a trajectory is excluded from all clusters, it does not necessarily indicate that it is an outlier. Furthermore, the problem of clustering has been shown to be more expensive than outlier detection [Yang et al. 2009]. That is, due to the interdependence among the data points, the cluster structure is more difficult to detect and in particular to update than maintaining the individual outlier points. Therefore, utilizing a clustering algorithm to detect outliers is not efficient in terms of CPU consumption.

3. NEIGHBOR-BASED TRAJECTORY OUTLIER

In this section, we first define the general notion of trajectory outliers in moving object streams. Three semantics for neighbor-based stream trajectory outlier detection are then introduced to discover such outliers.

The basic notion of capturing abnormal phenomena in data can be traced back to initial work by Hawkins [1980], which introduced the core principle still deployed for characterizing outliers. That is, an outlier is an observation that deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism. Following this notion, in a trajectory stream S, we consider a trajectory Tr as a stream trajectory outlier if it does not consistently have "enough" close neighbors in S, that is, if it is significantly different from the large majority of trajectories in the dataset S.

In the rest of this section, we present three trajectory outlier semantics: *PN-Outlier*, *TN-Outlier*, and *SN-Outlier*. These semantics, all based on the concept of trajectory

neighbor, that is, trajectories that are most similar to the trajectory in question, are referred to as a neighbor-based trajectory outlier. In Section 3.1, we first define the notion of a trajectory point, a stream trajectory, sliding window, and different variations of the trajectory neighbor concept that our neighbor-based trajectory outlier semantics are built on. Then, in Section 3.2, we introduce our taxonomy of outlier semantics in detail.

3.1. Notions

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 multidimensional data point p_i^j produced by the moving object o_i at time t_j is called a trajectory point of the trajectory Tr_i. We assume a minimal time granularity at which trajectory points are emitted. 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 a location-acquisition device. Instead, as an abstraction, it could correspond to any object that continuously produces distinct events along time. Correspondingly, the *trajectory point* is not necessarily a spatial position; rather, it could be any multidimensional coordinate for which each dimension corresponds to one of the attributes in the event produced by one object.

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

In this work, we use the periodic sliding window semantics as proposed by CQL [Arasu et al. 2006] for defining the substream of interest from the otherwise infinite trajectory data stream. In particular, an outlier detection request Q specifies a fixed window size w and slide size s for time-based windows. Count-based windows can be defined similarly. 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 respective 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 the 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 t_j. Without loss of generality, we utilize Euclidean distance as the distance measure, though any other distance measure could equally be plugged in.

Definition 3.1 (*Point Neighbor*). For two trajectory points p_m^j and p_n^j in the same timebin t_j, if $dist(p_m^j, p_n^j) \leq d$, 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 with regard to the distance threshold d comprise the **Point Neighbor Set** of p_n^j, denoted as $PN(p_n^j, d)$.

Definition 3.2 (*Trajectory Neighbor*). In window W_c, trajectory Tr_m is called a **trajectory neighbor** of Tr_n with regard to the given timebin count threshold thr_t, if and

[1]Trajectory data of moving objects that is not regular can also be handled; see discussion in Section 7.1.

only if there exist at least thr_t timebins in W_c such that p_m is a point neighbor of p_n at each of these thr_t timebins (with point neighbor defined in Definition 3.1).

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

Definition 3.3 (*Synchronized Trajectory Neighbors*). In window W_c, a set of trajectories $Tr_1, Tr_2, \ldots,$ and Tr_n are called **synchronized trajectory neighbors** of a given trajectory Tr_i with regard to the given timebin count threshold thr_t, if and only if at least thr_t timebins exist in W_c such that trajectory points $p_1^j, p_2^j, \ldots, p_n^j$ corresponding to $Tr_1, Tr_2, \ldots, Tr_n$ are point neighbors of trajectory point p_i^j corresponding to Tr_i in each timebin t_j of these thr_t timebins (with point neighbor as defined in Definition 3.1).

Intuitively, given a trajectory Tr_i, its synchronized trajectory neighbors (or in short *Sync neighbors*) Tr_m and Tr_n are also the *trajectory neighbors* of Tr_i by Definition 3.2, because Tr_i has at least thr_t point neighbors in both Tr_m and Tr_n. However, as an additional constraint, to be the *Sync neighbors* of Tr_i, the thr_t point neighbors in Tr_m and Tr_n must correspond to exactly the same thr_t timebins. In other words, the point neighbors in Tr_m and Tr_n have to be synchronized in time. That is why Tr_m and Tr_n are called the *synchronized* trajectory neighbors of Tr_i.

According to Definition 3.3, a trajectory Tr_i could have distinct groups of synchronized trajectory neighbors with regard to a different combination of thr_t timebins in W_c. We denote one particular *synchronized trajectory neighbor set* of Tr_i as $SN(Tr_i, d, thr_t)$ with regard to distance threshold d and timebin count threshold thr_t.

3.2. Trajectory Outlier Semantics

3.2.1. Point Neighbor–Based Trajectory Outlier.

Definition 3.4. Given a distance threshold d, a neighbor count threshold k, and timebin count threshold thr_t, the **point neighbor–based trajectory outlier semantics (PN-Outlier)** considers a trajectory Tr_i in the window W_c as an outlier of type **PN-Outlier**, if $|T| < thr_t$, where $T = \{t_j | \, |PN(p_i^j, d)| \geq k, \; t_j \in [W.T_{Start}, W.T_{End}]\}$. Otherwise, Tr_i is said to be 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* semantics is based on whether a trajectory has a sufficient number of neighboring timebins throughout the window. Intuitively, a normal inlier trajectory has at least a certain number of moving objects in its vicinity most of the time, that is, for at least thr_t timebins. 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 the other objects significantly.

Example 3.5. In Figure 1, let us consider Definition 3.4 when we assume that $k = 2$, $thr_t = 4$. The trajectory Tr_3 only has two neighboring timebins $\{t_1, \; t_4\}$, for which Tr_3 has more than k point neighbors. Thus, $|T| < 4$. Therefore, Tr_3 is a *PN-Outlier*. All other trajectories have at least thr_t neighboring timebins, thus are *PN-Inlier*s.

Applications. The *PN-Outlier* semantics effectively model the notion of outliers prevalent in many real-time applications. For example, in security surveillance, a foreign personnel may be considered as potentially unsafe if the person separates out from others for too much of the time during a visit. In a marathon race, a runner who runs alone without any other competitors around for long stretches of time warrants being tracked more closely. Reasons could range from the runner suffering from some

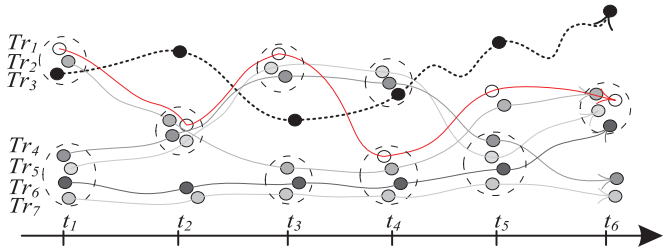


Fig. 1. Seven trajectories in a window W_c with size $w = 6$. A small circle represents a trajectory point. A large dashed oval indicates pairwise point neighbors. *PN-Outlier*: Tr_3. *PN-Inliers*: Tr_1, Tr_2, Tr_4, Tr_5,Tr_6, and Tr_7.

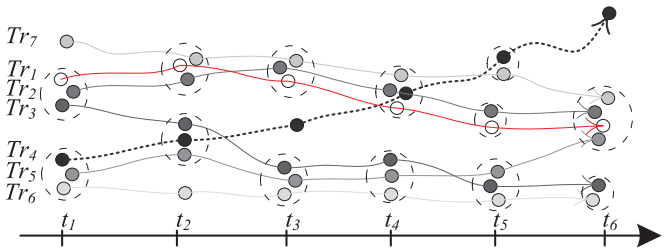


Fig. 2. Seven trajectories in a window with size $w = 6$, $k = 2$, $thr_t = 4$. *TN-Outlier*: Tr_4; *TN-Inlier*: Tr_1, Tr_2, Tr_3, Tr_5, Tr_6, Tr_7.

physical discomfort and possibly needing medical assistance to checking for possible cheating by taking shortcuts.

3.2.2. Trajectory Neighbor–Based Trajectory Outlier.

Definition 3.6. Given a distance threshold d, a neighbor count threshold k, and timebin count threshold thr_t, **trajectory neighbor–based trajectory outlier semantics (TN-Outlier)** considers a trajectory Tr_i in the window W_c as an outlier of type **TN-Outlier** if it has fewer than k trajectory neighbors $|TN(Tr_i, d, thr_t)| < k$ in W_c (with the trajectory neighbor concept as defined in Definition 3.2). Otherwise, the trajectory Tr_i is a **TN-Inlier**.

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

Example 3.7. Consider the trajectories in Figure 2. Suppose that $k = 2$, $thr_t = 4$. Then, Tr_4 is a *TN-Outlier* because it has no trajectory neighbors. Tr_1 and Tr_2 are trajectory neighbors of each other by Definition 3.2, 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 in at least 4 timebins. Since Tr_1, Tr_2, and Tr_7 each have k (2) trajectory neighbors, they are *TN-Inlier*s. Tr_3 has two trajectory neighbors Tr_5 and Tr_6. Thus, Tr_3 is a *TN-Inlier*.

Unlike the *PN-Outlier* semantics (Definition 3.4), for the *TN-Outlier*, it is no longer sufficient to be able to categorize a given trajectory as an outlier if it has a certain number of neighboring timebins. 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 earlier.

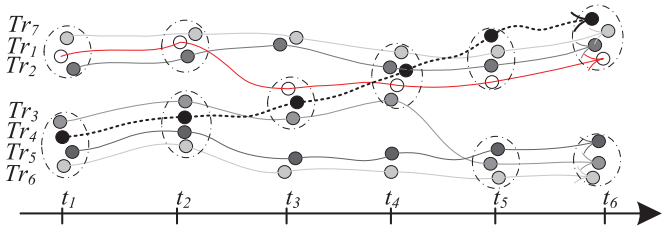


Fig. 3. Seven trajectories in a window with size $w = 6$, $k = 2$, $thr_t = 4$. *SN-Outlier*: Tr_4; *SN-Inlier*: Tr_1, Tr_2, Tr_3, Tr_5, Tr_6, Tr_7.

Applications. *TN-Outlier* fits many real-life applications. Consider traffic management applications [Yuan et al. 2011] in which 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 a similar speed with other cars. Similarly, in the stock ticker 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 have a similar price.

3.2.3. Synchronized Neighbor–Based Trajectory.

Definition 3.8. Given a distance threshold d, a neighbor count threshold k, and timebin count threshold thr_t, **Synchronized Neighbor–Based Trajectory Outlier semantics** (**SN-Outlier)** considers a trajectory Tr_i in a sliding window W_c as an outlier of type **SN-Outlier**, if there *does not* exist any synchronized trajectory neighbor set $SN(Tr_i, d, thr_t)$ of p_i such that $|SN(Tr_i, d, thr_t)| \geq k$ in W_c. Otherwise, the trajectory Tr_i is said to be an **SN-Inlier**.

Intuitively, to be an *SN-Inlier*, a trajectory Tr_i has to satisfy two conditions. First, similar to *TN-Inlier* in Definition 3.6, it has to have a sufficient number (k) of *trajectory neighbors* within the window W_c. In addition, it requires that at least k of these neighboring trajectories correspond to *synchronized trajectory neighbors* of Tr_i.

Example 3.9. Consider the trajectories in Figure 3. Suppose that $k = 2$, $thr_t = 4$, and Tr_1, Tr_2, and Tr_7 are pair-wise trajectory neighbors with each other by Definition 2. By Definitiion 3.3, Tr_1, Tr_2, and Tr_7 are also synchronized trajectory neighbors with each other with regard to a sufficient number ($\geq thr_t$) of timebins, because in W_c their trajectory points at each timebin but t_3 are also point neighbors of each other. Since Tr_1, Tr_2, and Tr_7 have at least 2 (k) synchronized trajectory neighbors, they are all *SN-Inliers*. The same situation also applies for Tr_3, Tr_5, and Tr_6.

Tr_4 has 2 ($= k$) trajectory neighbors $\{Tr_1, Tr_3\}$. Thus, it is a *TN-Inlier*. However, it would still be classified as an *SN-Outlier*, because $\{Tr_1, Tr_3\}$ are not *Sync* neighbors of Tr_4 . As shown in Figure 3, Tr_1 and Tr_3 both have point neighbors of Tr_4 only at 2 ($< thr_t$) timebins t_3 and t_4. This does not satisfy the *sync neighbors* definition as shown in Definition 3.3.

Applications. This *SN-Outlier* semantics fits real-life applications with strict synchronization requirements. For example, in military action, to understand the situation in a military battle field, the commander needs to be continuously aware of the moving groups of soldiers, vehicles, aircrafts, and other assets based on the objects' most recent positions (trajectories). The members of a tight military unit will be classified

as *SN-Outliers* if they do not continuously operate as a team. Similarly, biologists may track the moving trajectories of various species, such as the migration of the birds or whales, for their research purposes [Li et al. 2010; Tang et al. 2012]. The families of birds tend to move together in species migration. That is, most of the time, they would be expected to stay close enough to be neighbors. Otherwise, they will be considered as outliers (*SN-Outlier*) that are more likely to fall victims to predators or exhaustion.

3.3. Analysis of Trajectory Outlier Taxonomy

In essence, all three outlier types are defined based on either point-level or trajectory-level neighbor relationships. Thus, in general, they all fall into the class of neighbor-based trajectory outliers. However, unlike the traditional neighbor-based outlier [Knorr and Ng 1998], such neighbor-based trajectory outlier taxonomy not only relies on neighbor relationships in the spatial (multidimensional space) domain, but also on the behavior of the trajectories in the time domain. The rationale is that, in a trajectory stream, trajectory outliers usually do not only deviate from concurrent neighbors in their coordinate values in the multidimensional space, but their deviation also lasts for a certain time period.

Now, we analyze the relationships between the three types of trajectory outliers in the aforementioned taxonomy.

PROPERTY 1. *Given a trajectory outlier query Q applied to a window W_c over stream S, if a trajectory Tr_i is a PN-Outlier by Definition 3.4, then it must also be an SN-Outlier (Definition 3.8) with respect to the same parameter thresholds d, k, and thr_t.*

PROOF. By contradiction. If Tr_i is a *PN-Outlier*, then by Definition 3.4, Tr_i has less than thr_t neighboring timebins. Naturally, no set of k synchronized trajectory neighbors can exist that together participate in the same thr_t timebins of Tr_i. Otherwise, these thr_t timebins would also have been sufficient for Tr_i to be classified as *PN-Inlier*. Thus, Tr_i must also be an *SN-Outlier* by Definition 3.8. □

PROPERTY 2. *Given a trajectory outlier query Q applied to a window W_c over stream S, if a trajectory Tr_i is a TN-Outlier by Definition 3.6, then it must also be an SN-Outlier by Definition 3.8 with respect to the same parameter thresholds d, k, and thr_t.*

PROOF. By contradiction. Similarly, if Tr_i is a *TN-Outlier*, then by Definition 3.6, Tr_i has less than k trajectory neighbors. Therefore, its trajectory neighbors cannot form k synchronized trajectory neighbors, because that would require at least k neighbors in the same timebin for Tr_i. Thus, by Definition 3.8, Tr_i is also an *SN-Outlier*. □

Discussion. Two implications can be derived from these properties. First, the three outlier types are interdependent. That is, although they have distinct neighbor semantics, as required by different applications, all three rely on some inherent property of neighbor relationships in order to be classified as trajectory outliers. Second, *SN-Outlier* classifies more trajectories as outliers than either *PN-Outlier* or *TN-Outlier*, because *SN-Outlier* has stricter requirements for a trajectory to be considered a neighbor (*Sync* Neighbor) of another. Therefore, the outliers detected with the *PN-Outlier* or *TN-Outlier* definition are a subset of the *SN-Outlier* query result with regard to the same parameter setting. This property points us at optimization opportunities for the design of the *SN-Outlier* detector, as further discussed in Section 4. The relationships among these three neighbor-based trajectory outlier definitions are depicted in Figure 4.

4. THE BASIC OUTLIER DETECTOR

Given a trajectory stream S and the semantics of *PN-Outlier*, *TN-Outlier*, or *SN-Outlier*, we need to design algorithms to continuously detect and output the trajectory

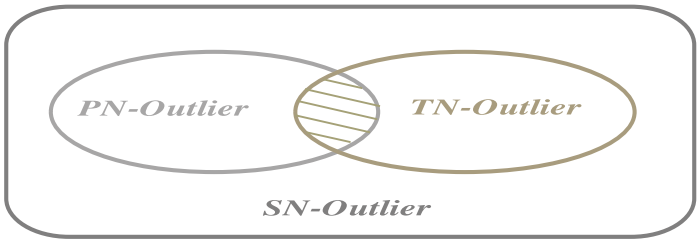
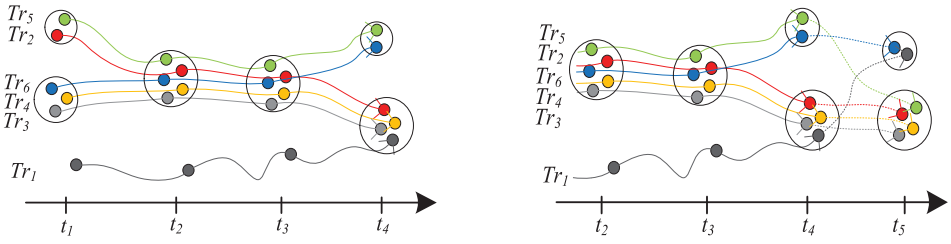


Fig. 4. Relationships of the three definitions.



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

Fig. 5. An example of two consecutive windows.

outliers in the latest window W_c as the window slides. As a foundation, we first design the INC algorithm that detects the three 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 avoids the repetition of redundant range query searches at the previously examined timebins, resulting in savings 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 Figure 6). 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 three outlier classes with the assistance of the just introduced $Tr_i.tlist$ and $Tr_i.NT$ structures. Figure 5 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 $thr_t = 3$, Figure 6 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 Figure 5(a), Tr_1

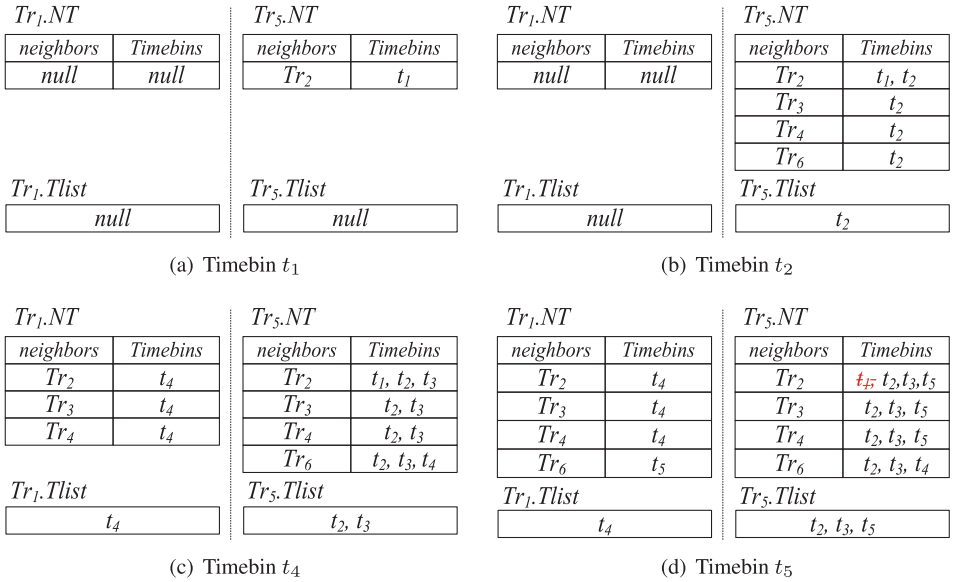


Fig. 6. Data structures of the INC algorithm.

does not have any point neighbor. Thus, both $Tr_1.NT$ and $Tr_1.tlist$ are null. Tr_5 instead acquires one point neighbor p_2^1 at timebin 1. Consequently, the record $<Tr_2, [t_1]>$ is created for p_2^1 and inserted into $Tr_5.NT$, as shown in Figure 6(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 timebin. After timebin t_4 is processed, the final neighboring information for W_1 is established, as shown in Figure 6(c).

Trajectory Outlier Detection. Next, using $Tr_i.tlist$ and $Tr_i.NT$, INC can quickly detect all classes of trajectory outliers. As shown in Figure 5(a), Tr_1 and Tr_5 can be immediately identified as *PN-Outlier*s because the cardinality of $Tr_1.tlist$ and $Tr_5.tlist$ (number of neighboring timebins) is smaller than thr_t (i.e., 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 $Tr_5.NT$.

Last, let us consider *SN-Outliers* of W_1. Since Tr_1 and Tr_5 are *PN-Outliers*, they are immediately recognized as *SN-Outliers* (see Property 1 in Section 3.3).

Data Structure Update. After the new timebin t_5 arrives, the window slides from $W1$ to $W2$ (Figure 5(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 computed using a range query search. The corresponding data structures are then updated (see Figure 6(d)).

Outlier Reexamination. Next, the status of all the trajectories will be reexamined 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 a new neighboring timebin at t_5. Thus, now, $|Tr_5.tlist| = thr_t$ (3).

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 Figure 6(d).

Concerning the *SN-Outlier* query, Tr_1 remains an *SN-Outlier* in $W2$ because it is a *PN-Outlier*. Since Tr_5 now is both a *PN-Inlier* and *TN-Inlier*, it cannot automatically be inferred to be an *SN-Outlier*. Instead, we now must employ one last check to verify the strictest condition imposed by the *SN-Outlier* semantics. That is, we have to verify if it has at least k synchronized trajectory neighbors with regard to thr_t timebins. Again, by examining the $Tr_5.NT$ table, we find that its trajectory neighbors $\{Tr_2, Tr_3, Tr_4\}$ indeed are also synchronized neighbors at the timebins t_2, t_3, and t_5 with Tr_5. Thus, Tr_5 is an *SN-Inlier* in W_2.

Algorithm 1 shows the INC algorithm for detecting the three classes of outliers. Given a trajectory Tr_i, $Tr_i.NT.update(Tr_j, W_c.T_{end})$ updates the $Tr_i.NT$ table of Tr_i with regard to trajectory Tr_j after receiving the new timebin $W_c.T_{end}$ of W_c. If the trajectory points of Tr_i and Tr_j in $W_c.T_{end}$ are point neighbors, the new timebin $W_c.T_{end}$ is inserted into the list of timebins corresponding to Tr_j in $Tr_i.NT$. On the other hand, $Tr_i.NT.remove(Tr_j, W_c.T_{start} - 1)$ removes the information corresponding to the expired timebin $W_c.T_{start} - 1$. Lines 8 to 11 show the update process $Tr_i.tlist$ that is specific to the *PN-Outlier*. If Tr_i has at least k point neighbors at timebin $W_c.T_{end}$, $W_c.T_{end}$ is inserted into $Tr_i.tlist$ as a new neighboring timebin (Lines 8–10). The expired timebin $W_c.T_{start} - 1$ will be removed from $Tr_i.tlist$ if it was a neighboring timebin of Tr_i.

INC then determines whether Tr_i is a *PN-Outlier* by examining the cardinality of $Tr_i.tlist$, namely, the number of Tr_i's neighboring timebins (Lines 12–14). Lines 15 to 17 examine whether Tr_i is a *TN-Outlier* by looking at the trajectory neighbors stored in $Tr_i.NT$. In particular, $getTN(Tr_i.NT, thr_t)$ discovers the trajectory neighbors of Tr_i that satisfy the constraint of thr_t timebins. More specifically, a trajectory Tr_j is the trajectory neighbor of Tr_i if the timebin list in $Tr_i.NT$ corresponding to Tr_j has at least thr_t elements. Finally, if Tr_i is neither *PN-Outlier* nor *TN-Outlier*, INC has to continue to verify whether Tr_i has k synchronized trajectory neighbors (denoted as $getSN(Tr_i.NT, thr_t)$). If not, then Tr_i is an *SN-Outlier*. $getSN(Tr_i.NT, thr_t)$ first finds all trajectory neighbors denoted as $\mathbb{TR}$ in $Tr_i.NT$, each of which has at least thr_t elements (timebins) in its corresponding list. Then, for each trajectory subset with k trajectories $\mathbb{TR}_k \subseteq \mathbb{TR}$, $getSN(Tr_i.TN, thr_t)$ examines whether the trajectories $\mathbb{TR}_k$ have the same thr_t timebins in their lists of $Tr_i.NT$. If so, trajectories in $\mathbb{TR}_k$ are synchronized trajectory neighbors of Tr_i. Tr_i is not an *SN-Outlier*. If no synchronized trajectory neighbor is found after all $\mathbb{TR}_k$s are examined, Tr_i is an *SN-Outlier*.

Complexity Analysis. The INC algorithm detects three 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 a *PN-Outlier* by checking the cardinality of each $Tr_i.tlist$. The cost is $O(n)$. Thus, 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 maintain and traverse the neighbor table $Tr_i.NT$ to discover trajectory neighbors. Its worst-case complexity is $O(2n^2)$. 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 the range query.

After acquiring all the trajectory neighbors and neighboring timebins, *SN-Outlier* detection still needs to examine whether the trajectory neighbors also form synchronized neighbors of Tr_i. Suppose that there are, on average, n' trajectory neighbors and w' neighboring timebins per trajectory. Thus, there are $C^k_{n'}$ combinations of k trajectory neighbors. $w'C^k_{n'}$ neighboring timebins may be examined in the worst case for verifying if each of the possible combinations corresponds to synchronized trajectory neighbors

ALGORITHM 1: INC algorithm(*PN-Outlier*, *TN-Outlier*, and *SN-Outlier*)

Require: Trajectory Set DB_{Tr}, the current window W_c, parameters: d, k, and thr_t.
Ensure: three types of outliers

```
1: for each Tr_i ∈ DB_Tr do
2:   for each Tr_j ∈ {DB_Tr − Tr_i} do
3:     if (dist(p_i^{W_c.T_end}, p_j^{W_c.T_end}) ≤ d) then
4:       Tr_i.NT.update(Tr_j, W_c.T_end);
5:       Tr_i.NT.remove(Tr_j, W_c.T_start − 1);
6:     end if
7:   end for
8:   if (|PN(p_i^{W_c.T_end}, d)| ≥ k) then
9:     Tr_i.tlist.add(W_c.T_end);
10:  end if
11:  Tr_i.tlist.remove(W_c.T_start − 1);
12:  //For PN-Outlier detection
13:  if (Tr_i.tlist.size() < thr_t) then
14:    Tr_i is PN-Outlier;
15:  end if
16:  //For TN-Outlier detection
17:  if (getTN(Tr_i.TN, thr_t).size() < k) then
18:    Tr_i is TN-Outlier;
19:  end if
20:  //For SN-Outlier detection
21:  if (Tr_i is PN-Outlier or Tr_i is TN-Outlier) then
22:    Tr_i is SN-Outlier;
23:  else if (getSN(Tr_i.TN, thr_t).size() < k) then
24:    Tr_i is SN-Outlier;
25:  end if
26: end for
```

per each trajectory. Therefore, its worst-case time complexity is $O(w'C_{n'}^{k}n)$. The overall complexity of *SN-Outlier* detection is $O(n^2) + O(w'C_{n'}^{k}n)$. Since w' and n' usually are much smaller than n, the cost of *SN-Outlier* detection is close to the cost of *TN-Outlier* detection, as confirmed in our experiments (Section 7.3). Therefore, like *TN-Outlier*, the time complexity of *SN-Outlier* is determined by the range query search and the lookup of its result.

5. 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, to a large degree, 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 presented earlier. To further drive down CPU and memory costs, we now present our *minimal examination* (*MEX*) optimized framework. By proposing three innovative optimization principles—*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.

5.1. Key Observations

Each trajectory in a window W is eventually classified 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 *nonneighbor evidence* might also be found to be sufficient in some cases 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 5.1 (*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 $(TR = \{Tr_1, \ldots Tr_x, \ldots Tr_m \,|Tr_x \in DB_{Tr}(1 \leq x \leq m)\}, T = \{t_1, \ldots t_j \ldots t_n \,|t_j \in [W_c.T_{Start}, W_c.T_{End}](1 \leq j \leq n)\})$ 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.

This structure of *minimal support* characterizes the minimal amount of evidence for identifying both *inliers* and *outliers*. This minimal support concept guides us to propose the *minimal support examination* principle (Section 5.2.1) 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 observation principle, *Predicted Support* in Lemma 5.2. This principle guides our MEX framework to discover the *best minimal support (MS)* for each trajectory.

Lemma 5.2 (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 Definition 5.1, 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 $Min(T)$ corresponds to the minimal timebin in T among all timebins of the minimal support set of Tr_i in the current window W_c.*

Proof. By Definitions 3.4 and 3.6, 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 (denoted as MS) of Tr_i expires, MS is still sufficient to determine the status of Tr_i in W_{c+1}. In Lemma 5.2, $W_{c+x}.T_{start} = Min(T)$. $Min(T)$ is the timebin arriving earliest in window W_c among all timebins in T. Therefore, $Min(T)$ will be the first timebin in T that expires when the stream moves from W_c to any future window. Since the starting timebin $W_{c+x}.T_{start}$ of W_{c+x} is equal to $Min(T)$, then W_{c+x} will be the last window in which all elements of MS are guaranteed to survive. However, (TR, T) continues to be minimal support for Tr_i in the windows from W_{c+1} to W_{c+x}. □

Lemma 5.2 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-triggered detection optimization* principle in Section 5.2.3. Second, the more windows a minimal support of Tr_i covers, the less reevaluation 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 Section 5.2.2.

5.2. Optimization Principles

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

For each principle, we first illustrate its impact on *PN-Outlier* and *TN-Outlier*. Given the relationships laid out in our taxonomy in Section 3.3, we then can leverage the results from these first two types of trajectory outliers to also support the third type of *SN-Outlier* detection. Thus, *SN-Outlier* equally benefits from these principles.

5.2.1. Minimal Support Examination Optimization. Leveraging the minimal support observation (Definition 5.1), the *minimal support examination* principle eliminates the complete and thus expensive point neighbor search for each trajectory adopted by the INC algorithm.

PRINCIPLE 1 (MINIMAL SUPPORT EXAMINATION OR MSE). *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 discovering only a small subset of its neighbors instead of searching through its complete neighborhood to classify all points with respect to their neighbor relationship with Tr_i. As shown in Corollaries 5.3 and 5.4, this principle is equally applicable to both *PN-Outlier* and *TN-Outlier* detection, although they each apply different concepts of neighbor semantics, respectively.

COROLLARY 5.3. *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 5.4. *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 5.3 and 5.4 directly follows from the definition of neighboring timebin (Definition 3.4) and *TN-Inlier* (Definition 3.6), respectively.

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

Definition 5.5 (*Examining operation*). 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 that we expect each trajectory may have, this examining operation is fundamentally more efficient than the full range query search. Here, we intuitively justify this observation.

In our neighbor-based outlier detection concept, a trajectory is an outlier if it has fewer than k neighbors. If we set k to be close to the average number of the neighbors

of all trajectories, then a large fraction of the trajectories will be identified as outliers. However, outliers by nature constitute only a small portion of the general stream data population. Otherwise, they would not be considered as outliers. Therefore, k has to be set much smaller than the average number of neighbors expected by each trajectory. Otherwise, the false positive will be exceedingly high. Therefore, our examining operation that *stops immediately* once finding k neighbors is fundamentally more efficient than a complete range query that aims to find *all* neighbors.

Furthermore, the MSE principle also guides us to optimize the acquisition process of the minimal neighbor support by leveraging the moving property of trajectory objects. For streaming trajectories, the objects continue to move and update their positions. However, the movement of objects tends to be regular and gradual rather than erratic and fast. Put differently, one object is likely to stay together with its neighbors throughout several consecutive timebins. Therefore, in general, the neighbors in the current window have a high probability to still be neighbors in the next window. The MSE principle leverages this observation and guides us to always first search neighbors for trajectory Tr_i in the trajectories that had been the neighbors of Tr_i instead of randomly picking and testing a neighbor candidate when the status reevaluation of Tr_i is necessary in a new window.

5.2.2. Time-Aware Examination Optimization. Our second optimization principle, called the *time-aware examination*, further optimizes the process of acquiring the timebin set T of the minimal support pair (TR, T).

Principle 2 (Time-aware Examination, or TAE). *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 unevaluated timebins until either neighbor evidence is found at thr_t timebins or nonneighbor evidence is confirmed for $(w - thr_t + 1)$ timebins.*

The TAE principle has two implications. First, the examining operation should evaluate the unevaluated 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-Outlier* and *TN-Outlier*.

For a *PN-Outlier* detection query with regard to 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 5.6. *Given a PN-Outlier detection query Q, if a trajectory Tr_i has already acquired thr_t neighboring timebins in the current window W_c, then Tr_i is a PN-Inlier in W_c as well as in the subsequent $(t_l\text{-}W_c.T_{Start})$ windows, where t_l is the oldest neighboring timebin acquired by Tr_i.*

Proof. First, by Definition 3.4, Tr_i is a *PN-Inlier* in W_c. Second, $\forall$ timebin $t_j \in T$, $t_j \geq t_l$. Therefore, the minimal timebin in T (*Min(T)*) is equal to t_l. By Lemma 5.2, Tr_i thus is also a *PN-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-Inlier* in the next $(t_l\text{-}W_c.T_{Start})$ windows. □

However, the status of Tr_i after window W_{c+x} (after t_l expires) is uncertain because the remaining evidence (thr_t-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 also effective for the *TN-Outlier* semantics (Lemma 5.7).

LEMMA 5.7. *Given a TN-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 timebin.*

PROOF. First, by Definition 3.2, Tr_i and Tr_m are trajectory neighbors in W_c. Second, all the thr_t neighboring timebins between Tr_i and Tr_m will survive in the future windows until timebin t_l expires. Therefore, Tr_i and Tr_m also can be safely classified as trajectory neighbors in the subsequent windows from W_{c+1} to W_{c+x}, where $W_{c+x}.T_{Start} = t_l$. Thus, Tr_i and Tr_m are guaranteed to be trajectory neighbors in the next (t_l-$W_c.T_{Start}$) windows. □

By Lemma 5.7, timebin t_l is subsequently called the **closed time** of Tr_i to be a trajectory neighbor of Tr_m.

LEMMA 5.8. *Given a TN-Outlier detection query Q, if Tr_i already has $(w - thr_t + 1)$ nonneighboring timebins with Tr_m in W_c, then Tr_i and Tr_m are not trajectory neighbors with each other in this window and the subsequent $(t_l - W_c.T_{start})$ windows, where t_l is their oldest nonneighboring timebin.*

PROOF. Since Tr_i already has $(w - thr_t + 1)$ nonneighbor timebins with Tr_m in the window W_c, then the number of remaining timebins is $thr_t - 1$. Thus, even if they all were neighboring timebins, Tr_i and Tr_m would not be trajectory neighbors in W_c. These dominating nonneighbor timebins also will survive in next windows until t_l expires. That is, Tr_i and Tr_m are not trajectory neighbors in the subsequent windows until t_l expires. Thus, the number of next windows also is (t_l-$W_c.T_{start}$). □

For Lemma 5.8, Tr_i and Tr_m would not be trajectory neighbors with each other until timebin t_l expires. Thus, we call t_l the **open time** of Tr_i with regard to Tr_m.

By Lemmas 5.6 to 5.8, t_l essentially determines the number of windows in which the evidence collected in the current window will survive. Therefore, the timebin search order of the TAE principle (from latest to earliest) effectively maximizes the reuse probability of the previously searched evidence.

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.

Finally, we describe the impact of the MSE and TAE principles on the *SN-Outlier*. Similar to INC, to detect the *SN-Outlier*, we first search the neighbor evidence for both the *PN-Outlier* and *TN-Outlier* types for each trajectory based on the MSE and TAE principles. Then, if a trajectory Tr_i is classified as both *PN-Inlier* and *TN-Inlier*, we proceed to search for neighbor evidence to attempt to acquire k synchronized trajectory neighbors of Tr_i.

LEMMA 5.9. *Given an SN-Outlier detection query Q, if in the current W_c, Tr_i has already acquired thr_t synchronized neighboring timebins with any k trajectory neighbors, then Tr_i is an SN-Inlier in W_c and in the next $(t_l - W_c.T_{start})$ windows, with t_l the oldest neighboring timebin.*

Lemma 5.9 follows directly from Definition 3.8 and Lemma 5.2. t_l is called the **closed time** for Tr_i to be an *SN-Inlier*.

LEMMA 5.10. *Given an SN-Outlier detection query Q, if in the current W_c, Tr_i has already discovered $w - thr_t + 1$ nonneighboring timebins, then Tr_i is an SN-Outlier in W_c and in the next $(t_l - W_c.T_{start})$ windows, with t_l the oldest timebin in these timebins.*

For Lemma 5.10, since Tr_i has already acquired $w - thr_t + 1$ timebins that are not neighboring timebins of Tr_i, k synchronized trajectory neighbors do not exist in the current window. Furthermore, Tr_i will not have k synchronized trajectory neighbors until at least timebin t_l expires. Thus, we call t_l the **open time** of Tr_i to be an *SN-Inlier*.

5.2.3. Lifetime-Triggered Detection Optimization. These two principles focus on how to optimize each single examining operation. To further reduce the computational costs, we now introduce another optimization principle regarding the minimization of the examining frequency, termed *lifetime-triggered detection optimization* (LTD). For that, we first define the *lifetime* concept of a trajectory.

Definition 5.11 (*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 $t_{lifetime}$ ($W_c.T_{Start} \leq t_{lifetime} \leq W_c.T_{End}$) expires according to the identified evidence, then $t_{lifetime}$ 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 three neighbor-based trajectory outlier definitions, respectively.

LEMMA 5.12. *Given a trajectory Tr_i, the lifetime of Tr_i to be a PN-inlier is the closed time of Tr_i in W_c.*

PROOF. Lemma 5.12 can be easily proven by Lemma 5.6. If Tr_i is a *PN-inlier*, Tr_i will remain to be inlier until its close time expires by Lemma 5.6. □

LEMMA 5.13. *Given a trajectory Tr_i, the lifetime of Tr_i to be a TN-inlier is $min\{Tr_j.closedTime | Tr_j \in TN\}$, where TN is the trajectory neighbor set of Tr_i in W_c.*

PROOF. If Tr_i is a *TN-inlier*, then by the MNP principle, Tr_i has acquired k trajectory neighbors. By Lemma 5.7, Tr_i will continue to be a neighbor with the trajectory Tr_j until its closed time $Tr_j.closeTime$ expires. Tr_i would not lose any neighbors until the minimal closed time of its k neighbors expires. Therefore, this minimal closed time is the lifetime of Tr_i. □

LEMMA 5.14. *Given a trajectory Tr_i, the lifetime of Tr_i to be an SN-inlier is the closed time of Tr_i in W_c.*

The proof naturally follows from the closed time concept of an *SN-inlier*.

Next, we define the LTD optimization principle based on this 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: $Tr_i.lifetime < W_c.T_{start}$.*

By the LTD principle, the status of a trajectory is reexamined only when its lifetime expires. This effectively transforms the **continuous** query execution into **lifetime-triggered** execution. Whenever a trajectory is being reexamined, the examining operation that incorporates both MSE and TAE optimizations can be exploited to reestablish 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* (LIFT).

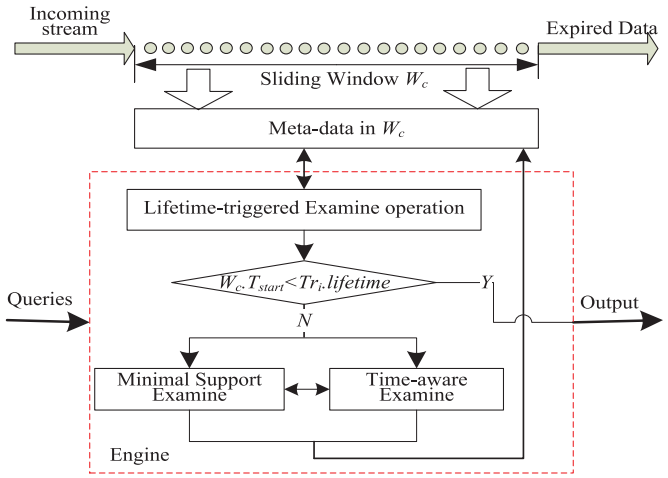


Fig. 7. Architecture of MEX framework.

5.3. Minimal Examination Framework

The high-level architecture of our MEX framework is depicted in Figure 7. Similar to the INC algorithm, the MEX framework utilizes the $Tr_i.tlist$ and $Tr_i.NT$ structures to store the metadata. 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 $W_c.T_{Start} \leq Tr_i.lifetime$. Once triggered, LIFT employs both of the MSE and TAE principles to establish the new minimal support. The status and 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.

6. MEX-BASED TRAJECTORY OUTLIER DETECTION ALGORITHMS

Data Structure Design. We first introduce the compact **Cyclic Bit Vector (CBit)** as core data structure for the implementation of the MEX-based trajectory outlier detection. Given a sliding window query Q with window size w, the CBit for each trajectory is a w-length bit vector. Each bit denotes the status of the trajectory at the corresponding timebin. For example, the neighboring timebin list $Tr_5.tlist$ of trajectory Tr_5 in Figure 5 is encoded as a 4-length bit vector. The first bit indicates the status of Tr_i for the first timebin of W_1, namely, $W_1.T_{Start}$. Correspondingly, the last bit indicates the status of Tr_i for the last timebin $W_1.T_{End}$. After the stream slides to the next window W_2, we simply shift the CBit with the new ending timebin, overwriting the old start timebin bit. The rest of the bits remain physically unchanged, although semantically they now represent different windows. The CBit structures of $Tr_5.tlist$ for windows W_1 and W_2 are shown in Figure 8, with 1 indicating a neighboring timebin and 0 a nonneighboring timebin.

6.1. The Optimized PN-Outlier Algorithm

Algorithm 2 shows the optimized *PN-Outlier* detection algorithm as an 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 nonneighboring and neighboring timebins of Tr_i, respectively. The LTD

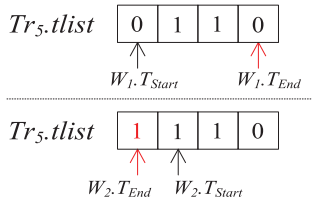


Fig. 8. An example of CBit.

ALGORITHM 2: PN-Opt Algorithm (*PN-Outlier* detection using MEX framework)

Require: Trajectory Set DB_{Tr}, the current window W_c, parameters: d, k, and thr_t.
Ensure: outliers

```
1: Get DB_lt ← (W_c.T_Start − 1).triggered;
2: for each Tr_i ∈ DB_lt do
3:   for each t_j ∈ Tr_i.untlist from W_c.T_End to head do
4:     if (true == Tr_i.LIFT.MSEInHistory(t_j)) then
5:       Tr_i.tlist.add(t_j);
6:     else if (true == Tr_i.LIFT.MSEInNewTrajectory(t_j)) then
7:       Tr_i.tlist.add(t_j);
8:     else
9:       Tr_i.ntlist.add(t_j);
10:    end if
11:    Tr_i.LIFT.TAE();
12:  end for
13:  if (Tr_i.tlist.size() < thr_t) then
14:    Tr_i.status=PN-Outlier;
15:  else
16:    Tr_i.status=PN-Inlier;
17:  end if
18:  Tr_i.updateLifetime();
19:  Lifetime.triggered.add(Tr_i);
20: end for
```

(lifetime-triggered detection) optimization is employed first (Line 1). That is, only the trajectories whose lifetime has expired are reexamined in the current window. For these triggered trajectories that must be examined, *PN-Opt* utilizes the MSE (minimal support examination) optimization principle to acquire new neighboring timebins (Lines 4–10). More specifically, *PN-Opt* first checks whether Tr_i can acquire k point neighbors in the new timebin by searching through the trajectories that have been neighbors with Tr_i in at least one other timebin, shown as $Tr_i.LIFT.MSEInHistory(t_j)$ (Line 4). The other remaining trajectory points will be tested only if Tr_i still has not acquired k point neighbors ($Tr_i.LIFT.MSEInNewTrajectory(t_j)$, Line 6). Then, the TAE principle is be utilized to determine whether the neighboring timebin search process should be terminated, namely, whether thr_t neighboring timebins or $w - thr_t + 1$ non-neighbor timebins have been acquired in the new window, denoted by $Tr_i.LIFT.TAE()$ (Lines 3 and 11). Finally, the lifetime of Tr_i, utilized by the LTD principle to determine whether the status of Tr_i should be reexamined, is updated (Lines 18 and 19). $Lifetime.triggered.add(Tr_i)$ adds Tr_i into the triggered list of the timebins of its lifetime.

6.2. The Optimized TN-Outlier Algorithm

Algorithm 3 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

ALGORITHM 3: TN-Opt Algorithm (*TN-Outlier* detection using MEX framework)

Require: Trajectory Set DB_{Tr}, the current window W_c, parameters: d, k, and thr_t.
Ensure: Outliers

```
1: Get DB_lt ← (W_c.T_Start − 1).triggered;
2: for each Tr_i ∈ DB_lt do
3:    for each Tr_m ∈ 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 ∈ (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=TN-Outlier;
14:      else
15:         Tr_i.status=TN-Inlier;
16:      end if
17:   end if
18:   Tr_i.updateLifetime();
19:   Lifetime.triggered.add(Tr_i);
20: end for
```

parts: *tlist*, *ntlist*, and *untlist*. *untlist* maintains the unchecked timebins between Tr_i and some trajectory Tr_j, while *ntlist* and *tlist* are the lists of nonneighboring and neighboring timebins between Tr_i and Tr_j. Similar to Algorithm 2, the *TN-Opt* algorithm also employs the LTD optimization 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.

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 thr_t timebins are found. By the MSE principle (Lines 5, 10), the trajectory neighbor search process stops immediately after Tr_i acquires sufficient (k) trajectory neighbors or the trajectories have been tested. Finally, the lifetime of Tr_i is updated (Lines 18, 19).

6.3. The Optimized SN-Outlier Algorithm

The optimized *SN-Outlier* detection algorithm called *SN-Opt* is shown in Algorithm 4. Similar to *TN-Opt*, *SN-Opt* also has four key data structures: $Tr_i.tlist$, $Tr_i.ntlist$, $Tr_i.untlist$, and $Tr_i.NT$. However, each element of $Tr_i.NT$ maintains only *tlist* and *ntlist*. As analyzed in Section 3, the *SN-Opt* algorithm first detects whether Tr_i is a *PN-Outlier* or *TN-Outlier* (Lines 3–5). If Tr_i is neither of them, then and only then we need to check whether the condition of $|\bigcap Tr_i.NT_k.value| \geq thr_t$ holds. That is, Tr_i is classified as an *SN-Inlier* if k synchronized trajectory neighbors are discovered. If Tr_i cannot be classified as an *SN-Inlier* by utilizing the neighbor information collected in the *PN-Outlier* and *TN-Outlier* detection procedure, we continue to apply the TAE principle to locate new neighboring timebins for Tr_i (Lines 7–9). If k synchronized trajectory neighbors are still not acquired after all timebins of the existing trajectory neighbors are traversed, we continue to locate new trajectory neighbors for Tr_i using the MSE optimization in Lines 10 to 12. The final status of Tr_i will be determined after all the trajectories have been evaluated (Lines 13 to 16) and the lifetime of Tr_i will be

ALGORITHM 4: SN-Opt Algorithm (*SN-Outlier* detection using MEX framework)

Require: Trajectory Set DB_{Tr}, the current window W_c, parameters: d, k, and thr_t.
Ensure: Outliers

```
1: Get DB_lt ← (W_c.T_Start − 1).triggered;
2: for each Tr_i ∈ DB_lt do
3:    PNOutlierDetection(Tr_i) and record Tr_i.NT;
4:    if Tr_i.status ≠ PN-Outlier then
5:       TNOutlierDetection(Tr_i) and record Tr_i.tlist, Tr_i.ntlist and Tr_i.untlist;
6:       if Tr_i.status ≠ TN-Outlier then
7:          while |&Tr_i.NT_k.tlst| < thr_t do
8:             check unevaluated timebins by Tr_i.LIFT.TAE();
9:          end while
10:         while |&Tr_i.NT_k.tlist| < thr_t do
11:            check unevaluated trajectories by Tr_i.LIFT.MSE();
12:         end while
13:         if |&Tr_i.NT_k.tlist| < thr_t then
14:            Tr_i.status=SN-Outlier;
15:         else
16:            Tr_i.status=SN-Inlier;
17:         end if
18:      end if
19:   end if
20:   Tr_i.updateLifetime();
21:   Lifetime.triggered.add(Tr_i); break;
22: end for
```

updated (Lines 20 and 21). As shown in Lines 7, 10, and 13, the synchronized trajectory neighbors can be efficiently located by using the $\&Tr_i.NT_k.tlist$ bit manipulation.

6.4. Complexity Analysis

PN-Opt detects outliers by conducting the LIFT operation at each new timebin per trajectory. Suppose that there are, on average, n' point neighbors out of n points at each timebin per trajectory and the outlier rate is, on average, α in each window. To confirm that a trajectory is a *PN-Outlier*, LIFT has to conduct a range query to prove that the new timebin is a nonneighboring timebin. Thus, the cost is $O(n)$. On the other hand, to confirm that a trajectory is a *PN-Inlier*, the cost is, on average, $O(\frac{kn}{n'})$ for acquiring k point neighbors in a timebin. Therefore, the overall cost is $O(\alpha n^2 + (1-\alpha)\frac{k}{n'}n^2)$. In general, the outlier rate α is very small and k is also set much smaller than the average number of neighbors, as described in Section 5.2.1. Thus, *PN-Opt* is fundamentally more efficient than INC, with complexity being $O(n^2)$.

TN-Opt detects the *TN-Outlier* by utilizing LIFT to discover k trajectory neighbors. Suppose that there are, on average, n' trajectory neighbors out of n trajectories per trajectory and outlier rate is, on average, α in each window. To confirm that a trajectory is a *TN-Outlier*, LIFT has to examine all n trajectories. In the worst case, the complexity is $O(wn)$. On the other hand, to confirm that a trajectory is a *TN-Inlier*, the complexity to acquire k trajectory neighbors is, on average, $O(w\frac{kn}{n'})$. Therefore, the overall complexity is $O(\alpha wn^2 + (1-\alpha)w\frac{k}{n'}n^2)$. Again, the outlier rate α is very small, and k is much smaller than the average number of the trajectory neighbors $n' < n$. Thus, *TN-Opt* is lightweight compared to INC, with INC's complexity $O(2n^2)$ for detecting *TN-Outlier*.

Similar to the *INC* algorithm, given a trajectory Tr_i, *SN-Opt* has to examine whether its trajectory neighbors also form synchronized neighbors. Suppose that there are, on average, n' trajectory neighbors and w' neighboring timebins per trajectory. $w'C^k_{n'}$ neighboring timebins might need to be examined to verify whether a combination of k

Table I. Description of Experimental Datasets

Datasets	# of objects	# of data points	# of dim	Duration of time	Area
Taxi	10357	15,000,000	2	7 days	Beijing
GMTI	150	100,000	3	6 hours	Military base
MOD	5000	2,400,000	2	500 timebins	San Francisco

trajectory neighbors is also a synchronized trajectory neighbor set of Tr_i. Therefore, its worst-case time complexity is $O(w' C_n^k n)$. The overall complexity of *SN-Opt* detection is $O(\alpha w n^2 + (1-\alpha) w \frac{k}{n'} n^2 + w' C_{n'}^k n)$.

7. EXPERIMENTAL EVALUATION

7.1. Experimental Setup

All algorithms are implemented in JAVA in the CHAOS stream engine [Gupta et al. 2009]. CHAOS supports multidimensional data and count-based/time-based sliding window streaming. The arrival rate of the streaming data also can be dynamic tuned in the CHAOS engine. In our experiments, the arrival rate is fixed as 500k tuples per second. Our experiments are performed on a PC with a 3.4G Hz Intel i7 processor and 6GB memory, which runs the Windows 7 OS.

Datasets. The *Taxi* dataset is the real GPS trajectory data generated by 10,357 taxis in a period from February 2 to February 8, 2008 in Beijing [Yuan et al. 2013, 2010]. The total number of points in this dataset is about 15 million. The average time interval between two points is around 177s. To model timebins, we interpolate the time granularity to 1minm per timebin.

The *GMTI* (Ground Moving Target Indicator) dataset [Entzminger et al. 1999] records the real-time trajectories of 150 moving objects gathered by 24 different data ground stations or aircraft in 6h. 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 3 dimensions of GMTI while detecting outliers based on the targets' latitude, longitude, and altitude.

We use the Moving Objects Database (MOD) generated by Thomas Brinkhoff's network-based moving object generator [Brinkhoff 2002] using real road networks. This dataset contains 5,000 trajectories of moving objects. Each trajectory has 500 timebins. This dataset models the traffic condition in the San Francisco Bay Area.

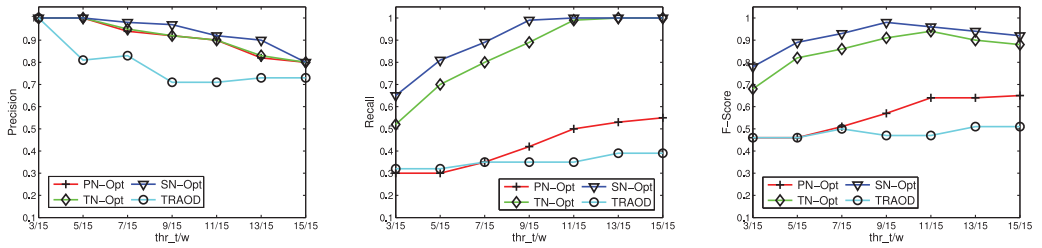
Table I shows the attributes of our three datasets. For one trajectory, multiple values are collected in one timebin. We currently adopt the latest value, though another methodology could be chosen, such as the average among these points. This does not affect our proposed methodology. On the other hand, if no value is observed in one timebin t_i, we utilize the values in the last two timebins t_{i-2} and t_{i-1}, and estimate the value of timebin t_i by assuming that the moving object follows a linear model. This method effectively supports trajectory data with different sampling rates. Again, in the future, other methods for dealing with missed values could easily be plugged in and would be orthogonal to our methodology.

Metrics and Measurements. We evaluate both (1) the effectiveness of our outlier definitions and (2) the efficiency of our MEX outlier detection algorithms.

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

$$Precision = \frac{|R_o \cap D_o|}{|D_o|}, Recall = \frac{|R_o \cap D_o|}{|R_o|}, F\text{-}Score = 2 * \frac{precision * recall}{precision + recall},$$

where R_o denotes the set of annotated outliers in a dataset, that is, the real outliers that correspond to our ground truth and D_o denotes the outliers detected by our algorithms.



(a) Precision (b) Recall (c) F-Score

Fig. 9. Effectiveness with regard to timebin count threshold thr_t (GMTI data).

For the *efficiency* evaluation, we measure two metrics common for stream systems: CPU processing time averaged per window and peak memory consumption.

7.2. Effectiveness Evaluation

We evaluate the effectiveness of the new proposed outlier definitions compared with the state-of-the-art TRAOD [Lee et al. 2008] by measuring the *precision*, *recall*, and *F-Score* on the GMTI, Taxi, and MOD datasets.

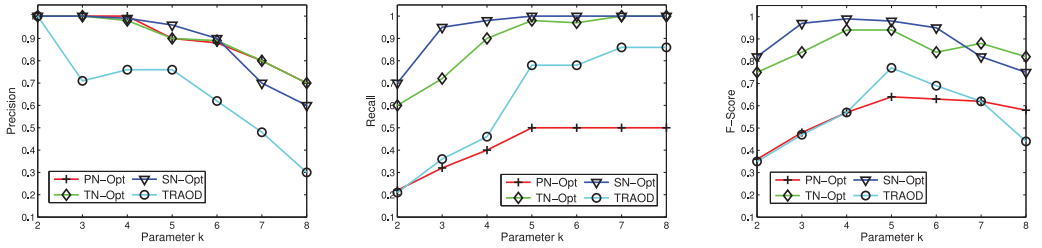
For the GMTI dataset, we use the outlier set manually labeled by the experts as the ground truth R_o. In this GMTI dataset, the moving objects (vehicles, helicopters, or soldiers) are divided into 3 units. The members of each unit are expected to operate as a team. If they do not continuously stay close to other team members, they will be labeled as outliers.

For the Taxi dataset, the ground truth outlier set R_o is produced by a user study. In this user study, 100 sets of trajectories were selected from the Taxi dataset. Each set contains 10 trajectories within 30 consecutive 6min windows. We invited 50 users from both Worcester Polytechnic Institute (WPI) and University of Science and Technology Beijing (USTB) as participants. The users, including both undergraduate and graduate students, were divided into 5 groups. Each group was assigned 10 sets of trajectories. The participants were asked to mark trajectories in each set that they believe are most likely outliers. Each trajectory marked by at least 5 users is labeled as a "real" outlier.

For the MOD dataset, we label outlier moving objects by considering their location and velocity attributes. Thus, the labeled outliers (the outlier cars) include two categories: vehicles that always drive at the urban edges and vehicles that drive in the downtown area but with extremely high or low speed compared to normal city traffic.

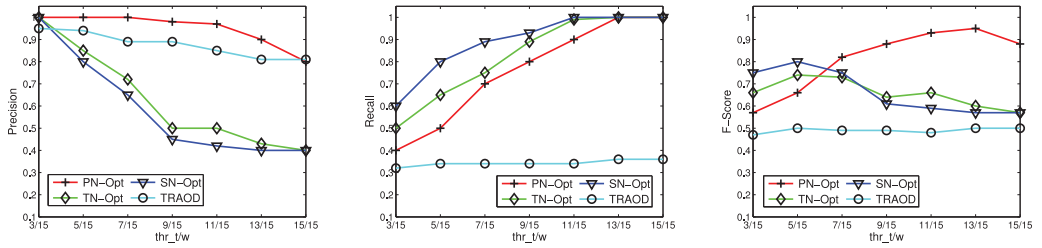
For all effectiveness experiments, we vary the thresholds k and thr_t to investigate how *Precision*, *Recall*, and *F-Score* are impacted. The d threshold is fixed at 200 meters, 200 meters and 300 meters for MOD, GMTI, and Taxi data, respectively. The window size is fixed to 15 for the experiments with all three datasets.

GMTI Data. The results for the GMTI data are shown in Figures 9 and 10. We fix k to 4 and thr_t to 10 by default when varying each one of the two thresholds. In Figures 9(a) and 10(a), the *Precision* of *PN-Outlier*, *TN-Outlier*, and *SN-Outlier* is nearly 100% once the parameters k and thr_t fall in a rather large range ($thr_t \leq 9$ and $k \leq 5$). All three are superior to TRAOD with respect to *Precision* in all tested cases. On the other hand, the *Recall* of both *PN-Outlier* and TRAOD (Figure 9(b) and Figure 10(b)) are much worse than *TN-Outlier* and *SN-Outlier*. The reason is as follows. Since *PN-Outlier* would not classify an object as outlier if there are a sufficient number of moving objects in its vicinity, sometimes *PN-Outlier* might fail to locate the object separated from its own unit. TRAOD only discovers the outlier t-partitions that are distant from other line segments of all trajectories in the spatial domain. However, the distance function for line segments proposed in TRAOD is not well suitable for the scenario in



(a) Precision (b) Recall (c) F-Score

Fig. 10. Effectiveness with regard to neighbor count threshold k (GMTI data).



(a) Precision (b) Recall (c) F-Score

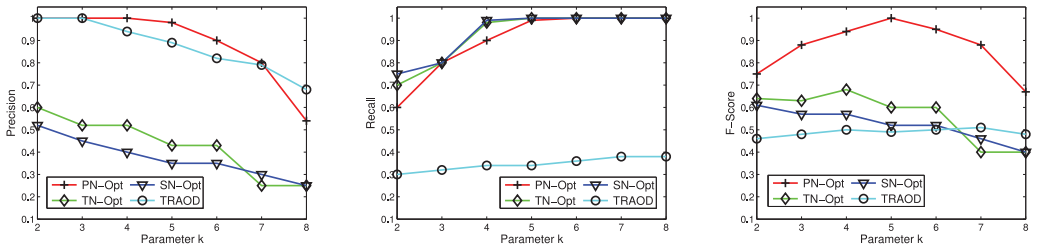
Fig. 11. Effectiveness with regard to timebin count threshold thr_t (Taxi data).

which objects may move in any direction because it does not satisfy the rule of triangle inequality. This is also reflected by the *Precision* of TRAOD in Figures 9(a) and 10(a).

As depicted in Figures 9(c) and 10(c), *SN-Outlier* again outperforms *TN-Outlier*, *PN-Outlier*, and TRAOD in *F-Score* in most tested cases. Although the *Recall* of *TN-Outlier* is also good and robust, as shown in Figure 9(b), the parameter range that leads to good recall does not overlap with the parameter range that produces good precision results (Figure 9(a)). This is because, to classify one trajectory as normal (inlier), *TN-Outlier* does not require the neighbors of that trajectory to move in a synchronized way, that is, at the same pace. Therefore, *TN-Outlier* might fail to recognize the moving objects that occasionally travel together with the members of different teams instead of strictly taking the expected route of their own team. For the *SN-Outlier*, a large range of parameter settings produce both good precision and recall. The reason is that *SN-Outlier* will classify all trajectories as outliers if they do not have neighbors that consistently follow a similar route, even if they happen to stay close to others at times. Therefore, *SN-Outlier* fits the scenario of the GMTI data among all contenders.

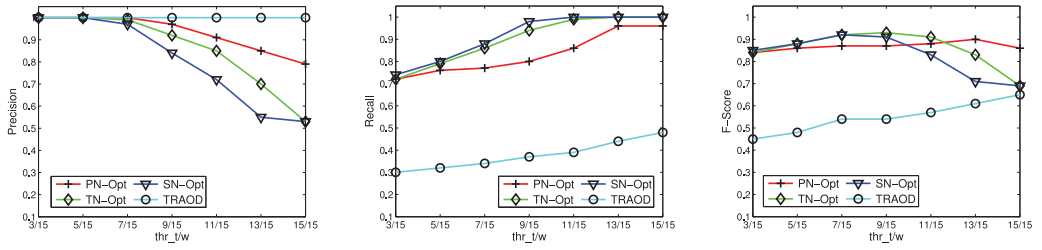
Taxi Data. We also investigate the effectiveness of *PN-Outlier*, *TN-Outlier*, *SN-Outlier*, and TRAOD approaches for analyzing outliers in the Taxi data. We fix k to 5 and thr_t to 11 when varying one of them. As shown in Figures 11(a), 11(b), 12(a), and 12(b), *PN-Outlier* shows nearly 100% *Precision* and *Recall* for a large range of parameters k and thr_t. However, the *Precision* of both *TN-Outlier* and *SN-Outlier* is significantly worse than that of *PN-Outlier*, although their *Recall* is as good as that of *PN-Outlier*.

The spatial outliers discovered by TRAOD are most likely a subset of those detected by *PN-Outlier* in this traffic scenario. That is why the *Precision* of TRAOD is as good as that of *PN-Outlier*, but the *Recall* of TRAOD is much worse than that of *PN-Outlier*, meaning it cannot identify all outliers. In this user study, we observe that the participants tend to classify a taxi as an outlier if it always moves alone, because intuitively taxis do not necessarily move together with others as a group. Therefore, the behavior



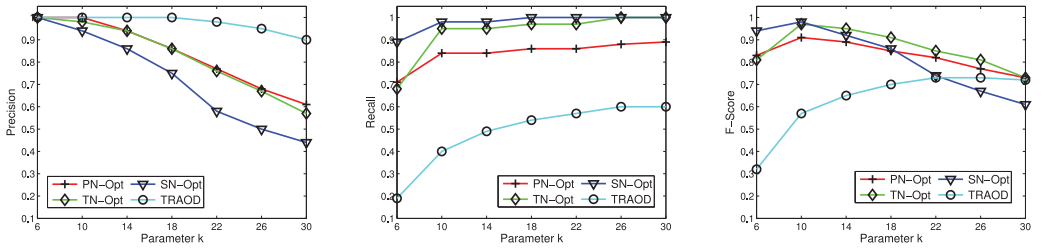
(a) Precision (b) Recall (c) F-Score

Fig. 12. Effectiveness with regard to neighbor count threshold k (Taxi data).



(a) Precision (b) Recall (c) F-Score

Fig. 13. Effectiveness with regard to timebin count threshold thr_t (MOD data).



(a) Precision (b) Recall (c) F-Score

Fig. 14. Effectiveness with regard to neighbor count threshold k (MOD data).

of a taxi driver will be considered as abnormal by the participants if it tends to operate in areas that other drivers rarely visit. Put differently, those taxis are isolated from others most of the time. As shown in Figures 11(c) and 12(c), the *F-Score* of *PN-Outlier* is much better than the other three definitions in a rather large parameter range. That is, this scenario fits the *PN-Outlier* model better than *TN-Outlier* and *SN-Outlier*, since the later *TN-Outlier* and *SN-Outlier* tend to misclassify the taxis as outliers if they lack consistent companions.

MOD Data. We also evaluate the effectiveness of the three outlier definitions in comparison to TRAOD using the MOD data. We fix k to 10 and thr_t to 9 for the experiments unless either k or thr_t is explicitly varied. As shown in Figures 13(a), 13(b), 14(a), and 14(b), *TN-Outlier* shows between 90% to 100% *Recall*, very good *Precision*, and thus consistently a good *F-Score* in most tested cases. The reason is as follows. In the MOD traffic scenario, the vehicles usually drive on the main streets with several cars in front of or behind them. During a certain time interval, the cars with normal velocity consistently travel with other cars that move at a similar speed. However, these cars might not strictly move together as a motorcade. Although the cars with

higher or lower speed on the main road also have neighboring cars, their neighboring cars may change by overtaking or by being overtaken continuously. Thus, this scenario fits *TN-Outlier* the best.

On the other hand, although the *Precision* of *PN-Outlier* is also fairly robust, as shown in Figure 14(a), its *Recall* is barely satisfactory compared to *TN-Outlier*, as shown in Figure 14(b). The reason is that *PN-Outlier* misses the "abnormal" cars that happen to be moving on the main road together with other vehicles, yet they are speeding or are driving at a low speed. TRAOD suffers from the same problem, discovering only specific spatially outlier t-partition trajectories. This causes low *Recall* but high *Precision*, meaning that these trajectories detected as outliers are true outliers. In contrast, *SN-Outlier* instead exhibits good results for *Recall* (Figures 13(b) and 14(b)) similar to *TN-Outlier*. However, its *Precision* is poor (Figures 13(a) and 14(a)). *SN-Outlier* classifies the vehicles as outliers if they do not drive together as a convoy. However, unlike the military units in GMTI data, in fact, the cars on the road are not expected to move strictly together. Therefore, *SN-Outlier* tends to classify too many vehicles as outliers due to its overstrict neighbor criteria.

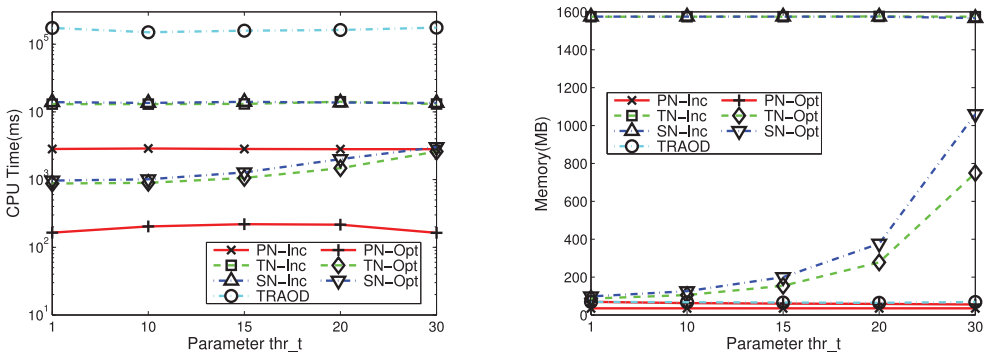
Summary. In summary, this empirical study confirms the effectiveness of our new proposed *PN-Outlier*, *TN-Outlier*, and *SN-Outlier* definitions in capturing distinct cases of moving object outliers. Furthermore, it shows that the three definitions tend to cover distinct application categories based on their respective semantics of what it means to be considered as neighbors.

7.3. Efficiency Evaluation

Next, we evaluate the efficiency of our proposed outlier detection algorithms using the Taxi data − the largest among our datasets. The performance of the MEX algorithms is compared with our base INC algorithms (Section 4) and the state-of-the-art TRAOD [Lee et al. 2008]. The INC algorithms support all classes of our proposed outlier semantics and effectively leverage the overlap of sliding windows. We denote the INC solution for *PN-Outlier*, *TN-Outlier*, and *SN-Outlier* detection as *PN-INC*, *TN-INC*, and *SN-INC*, and the *MEX*-based solution as *PN-Opt*, *TN-Opt*, and *SN-Opt*, respectively. TRAOD was proposed to detect outliers in a static trajectory database. In this revision, we extend it to also support streaming trajectory data. More specifically, we implement TRAOD to function in sliding window environments. After each slide, TRAOD considers all trajectories of moving objects in the current window as static trajectories. When the window slides, after removing the expired data and including the new arrivals, TRAOD is applied again to update the outlier status of each trajectory in the new window. We vary the most important parameters—the timebin count threshold thr_t, neighbor count threshold k, and the number of trajectories n—to (1) assess their impact on our *MEX* framework versus the INC baseline and TRAOD and (2) evaluate the impact of the parameter variations on the performance of each method.

7.3.1. Varying Timebin Count Threshold thr_t.

CPU Processing Time. 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 (i.e., one trajectory Tr_i is inlier, e.g., if *only in 1* of w timebins tr_i has the needed k 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 200m. As expected, our MEX-based algorithms are superior to the corresponding INC-based solutions with regard to the CPU time in all cases (Figure 15). Among all these algorithms, TRAOD is the slowest one, almost 118 times slower than *SN-Opt*, on average. This is because TRAOD has to compute distances between new t-partitions and all t-partitions of other moving objects to detect outlier subtrajectories. *PN-Opt* is 117 times faster than



(a) CPU time (log scale) (b) Memory

Fig. 15. Varying thr_t on taxi data.

PN-INC (Figure 15(a)). When thr_t is equal to the full window, the outlier rates of the three definitions are at their highest: 5%, 20%, and 20%, respectively. However, even in these cases, the optimized algorithms still outperform their corresponding counterparts 31-, 5- and 4-fold, respectively.

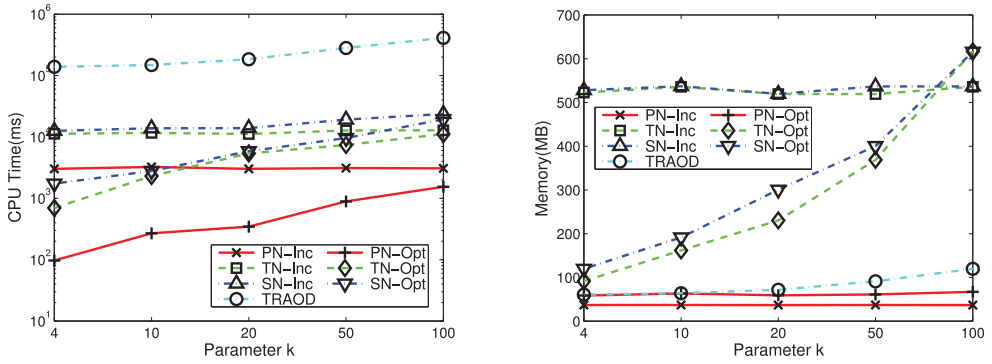
We also notice that the INC algorithms are not sensitive to thr_t since INC always tests all timebins of each trajectory, whereas by employing the TAE optimization principle (Section 5.2.2), MEX significantly reduces the distance calculation by only acquiring the minimal yet sufficient evidence to prove the outlier status of each trajectory.

Memory Consumption. We see similar positive trends also for the memory usage (see Figure 15(b)). That is, *TN-Opt* and *SN-Opt* each use, on average, 21% and 25% of the memory consumed by their counterpart *INC* algorithms, respectively. This can be explained by the fact that the MEX framework only maintains k trajectory neighbors for *TN-Outlier* and *SN-Outlier*, while the INC solution aggressively stores all neighbor information. As expected, the memory consumption of *TN-Opt* and *SN-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 Figure 15(b), the memory usage of *PN-Opt*, *PN-INC*, and TRAOD is very small. *PN-Opt* and TRAOD use a little more memory than *PN-INC*. This is because TRAOD also stores all neighboring t-partitions for each t-partition, which are used to detect if the t-partition is an outlier. For each trajectory Tr_i, *PN-Opt* stores the trajectories that have point neighbors with Tr_i in the history to reduce the trajectory search scope for acquiring new neighboring timebins. However, this extra memory leads to huge gains in CPU processing resources (around 30 times faster than *PN-INC* in the worst case, as shown in Figure 15(a)).

7.3.2. Varying Neighbor Count Threshold k. Next, we evaluate the performance of the seven 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 Processing Time. Figure 16(a) demonstrates that the three MEX-based algorithms clearly outperform their INC counterparts and TRAOD. The three MEX-based algorithms save, on average, 91%, 56%, and 57% of CPU time compared to their corresponding INC solution. Again, all proposed algorithms consistently outperform TRAOD around 2 orders of magnitude in CPU time. As k increases, the CPU time of the MEX-based solutions increases linearly since more neighbors have to be acquired to



(a) CPU time (log scale) (b) Memory

Fig. 16. Varying k on taxi data.

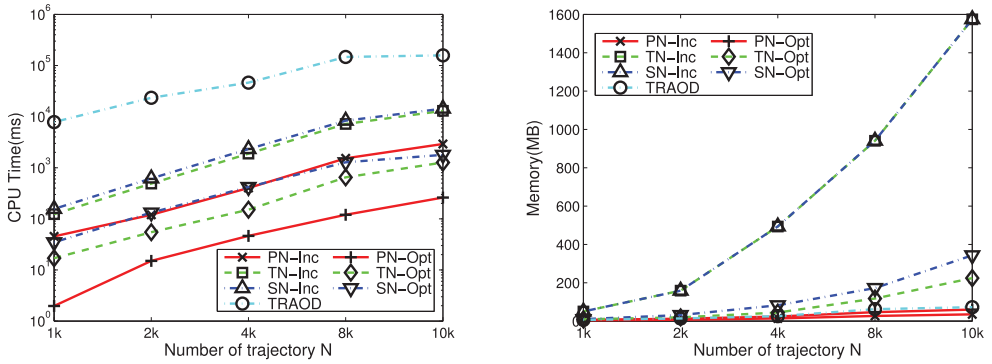
determine the status of a given trajectory. For *PN-INC* and *TN-INC* instead we observe no sensitivity when varying k. This is because these two INC solutions determine the status of Tr_i always by first acquiring all neighbors with a complete range query search independent from how large or small k is, while *SN-INC* consumes a little more CPU time as k increases since it needs more time to find the k synchronized trajectory neighbors, as described in the section on complexity analysis (Section 4). We note that 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, 90%, and thus unrealistic as an indicator for anomaly in this dataset.

Memory Consumption. *TN-Opt* and *SN-Opt* only use, on average, 55% and 60%, respectively, of the memory compared to the INC-based algorithms (Figure 16(b)). As k increases, they need to store more trajectory neighbors. Thus, their memory consumption increases. As k increases to 100, the maximum value that we work with, *TN-Opt* and *SN-Opt* use more memory than the INC-based solution. Again, this corresponds to an unrealistic setting in this dataset that we do not expect to see in practice, as surely 90% of all data would not be considered as an "exception" (outlier) but rather the norm.

Similar to the varying timebin count threshold experiments, TRAOD, *PN-Opt*, and *PN-INC* use much less memory than *TN-Outlier* and *SN-Outlier* detection algorithms. However, as k increases, TRAOD also has to store more neighboring trajectories for each t-partition. Thus, its memory consumption increases somewhat. *PN-Opt* and *PN-INC* are not sensitive to k with regard to memory, since they do not keep point neighbors for each trajectory.

7.3.3. Varying Number of Trajectories n. We evaluate the scalability of our algorithms in terms of the number of trajectories that they can simultaneously handle. In this experiment, we randomly select from 1k up to 10k trajectories from the Taxi dataset. We fix the window size to 30, k to 4, and 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 200m to 300m.

CPU Processing Time. As shown in Figure 17(a), the MEX algorithms exhibit much better scalability than their INC-based counterparts and TRAOD. As the number of trajectories increases, as expected, all algorithms require more time to process this larger number of trajectories. However, the MEX algorithms increasingly outperform the INC-based solutions as the number of trajectories increases because, unlike the INC solution, MEX avoids expensive range queries for the needed neighbor searches. Similarly,



(a) CPU time (log scale) (b) Memory

Fig. 17. Varying number n of trajectories on Taxi data.

as the number of trajectories increases, TRAOD spends more time on computing distances between t-partitions to update neighboring trajectories for each t-partition.

Memory Consumption. As shown in Figure 17(b), the memory consumption of all algorithms increases as the number of trajectories increases because more trajectory information must be stored. However, *TN-Opt* and *SN-Opt* both save more memory than *TN-INC* and *SN-INC* as n increases, while the memory usage of *PN-Opt*, *PN-INC*, and TRAOD remains consistently small.

8. CONCLUSIONS

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

In the future, we will investigate whether other distance measures, such as Dynamic Time Warping, would fit some categories of stream applications better than Euclidean distance in detecting moving-object outliers. We will also investigate the opportunities to further scale the detection of trajectory outliers by leveraging the distributed computation power of computer clusters. Moreover, we will also explore the new outlier approach that is able to detect outliers that are less similar to the majority trajectories to further enrich the literature of stream trajectory outlier detection.

REFERENCES

Arvind Arasu, Shivnath Babu, and Jennifer Widom. 2006. The CQL continuous query language: Semantic foundations and query execution. *The VLDB Journal* 15, 2, 121–142.

Thomas Brinkhoff. 2002. A framework for generating network-based moving objects. *GeoInformatica* 6, 2, 153–180.

Yingyi Bu, Lei Chen, Ada Wai-Chee Fu, and Dawei Liu. 2009. Efficient anomaly monitoring over moving object trajectory streams. In *Proceedings of SIGKDD*. ACM, 159–168.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *Computer Surveys* 41, 3, Article 15.

John N. Entzminger, Charles A. Fowler, and William J. Kenneally. 1999. JointSTARS and GMTI: Past, present and future. *IEEE Transactions on Aerospace and Electronic Systems* 35, 2, 748–761.

Yong Ge, Hui Xiong, Chuanren Liu, and Zhi-Hua Zhou. 2011. A taxi driving fraud detection system. In *Proceedings of ICDM*. IEEE, 181–190.

Yong Ge, Hui Xiong, Zhi-Hua Zhou, Hasan Ozdemir, Jannite Yu, and K. C. Lee. 2010. TOP-EYE: Top-k evolving trajectory outlier detection. In *Proceedings of CIKM*. ACM, 1733–1736.

Chetan Gupta, Song Wang, Ismail Ari, and Ming Hao. 2009. CHAOS: A data stream analysis architecture for enterprise applications. In *Proceedings of CEC*. IEEE, 33–40.

Douglas M. Hawkins. 1980. *Identification of Outliers*. Springer, New York, NY.

Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. 2008. Discovery of convoys in trajectory databases. In *Proceedings of VLDB*. ACM, 1457–1459.

Edwin M. Knorr and Raymond T. Ng. 1998. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of VLDB*. 392–403.

Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. 2000. Distance-based outliers: Algorithms and applications. *The VLDB Journal* 8, 3–4, 237–253.

Jae-Gil Lee and Jiawei Han. 2007. Trajectory clustering: A partition and group framework. In *Proceedings of SIGMOD*. ACM, 593–604.

Jae-Gil Lee, Jiawei Han, and Xiaolei Li. 2008. Trajectory outlier detection: A partition-and-detect framework. In *Proceedings of ICDE*. IEEE, 140–149.

Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. 2007. ROAM: Rule- and motif-based anomaly detection in massive moving object data sets. In *Proceedings of SDM*. SIAM, 273–284.

Zhuihui Li, Bolin Ding, Jiawei Han, and Roland Kays. 2010. Swarm: Mining relaxed temporal moving object clusters. In *Proceedings of VLDB*. 723–734.

Wei Liu, Yu Zheng, and Sanjay Chawla. 2011. Discovering spatio-temporal causal interactions in traffic data streams. In *Proceedings of SIGKDD*. ACM, 1010–1018.

Lu An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Chih-Chieh Hung, and Wen-Chih Peng. 2012. On discovery of traveling companions from streaming trajectories. In *Proceedings of ICDE*. 186–197.

Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. 2009. Neighbor-based pattern detection for windows over streaming data. In *International Conference on Extending Database Technology: Advances in Database Technology*. 529–540.

Yanwei Yu, Lei Cao, Elke A. Rundensteiner, and Qin Wang. 2014. Detecting moving object outliers in massive-scale trajectory streams. In *Proceedings of KDD*. ACM, 422–431.

Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2011. Driving with knowledge from the physical world. In *KDD*. 316–324.

Jing Yuan, Yu Zheng, Xing Xie, and Guangzhong Sun. 2013. T-Drive: Enhancing driving directions with taxi drivers' intelligence. *Transactions on Knowledge and Data Engineering* 25, 1, 220–232.

Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: Driving directions based on taxi trajectories. In *GIS*. 99–108.

Daqing Zhang, Nan Li, Zhi-Hua Zhou, Chao Chen, Lin Sun, and Shijian Li. 2011. iBAT: Detecting anomalous taxi trajectories from GPS traces. In *Proceedings of UbiComp*. ACM, 99–108.

Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *Proceedings of ICDE*. IEEE, 186–197.

Kai Zheng, Yu Zheng, Nicholas Jing Yuan, Shuo Shang, and Xiaofang Zhou. 2014. Online discovery of gathering patterns over trajectories. *IEEE Transactions on Knowledge and Data Engineering* 26, 8, 242–253.

Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin* 33, 2, 32–40.

Received August 2015; revised October 2016; accepted October 2016