Efficient Anomaly Monitoring Over Moving Object Trajectory Streams

Yingyi Bu[1] Lei Chen[2] Ada Wai-Chee Fu[1] Dawei Liu[1]

[1] The Chinese University of Hong Kong
yybu,adafu,dwliu@cse.cuhk.edu.hk

[2] Hong Kong University of Science and Technology
leichen@cse.ust.hk

Lately there exist increasing demands for online abnormality monitoring over trajectory streams, which are obtained from moving object tracking devices. This problem is challenging due to the requirement of high speed data processing within limited space cost. In this paper, we present a novel framework for monitoring anomalies over continuous trajectory streams. First, we illustrate the importance of distance-based anomaly monitoring over moving object trajectories. Then, we utilize the local continuity characteristics of trajectories to build *local clusters* upon trajectory streams and monitor anomalies via efficient pruning strategies. Finally, we propose a *piecewise metric index structure* to reschedule the joining order of local clusters to further reduce the time cost. Our extensive experiments demonstrate the effectiveness and efficiency of our methods.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Multimedia Databases*; H.2.8 [**Database Management**]: Database Application—*Data Mining*

General Terms

Algorithm, Design, Experimentation

1. INTRODUCTION

Recently, trajectory mining has attracted much attention due to its wide applications, especially in context-aware computing environment. Many researchers have worked on trajectory clustering [23] and classification [13]. Sometimes abnormal trajectories tend to carry critical information of potential problems which require immediate attention and need be resolved at an early stage. There are many applications that require real-time monitoring of abnormal sequential patterns over streaming trajectories. Examples include elder care, child custody, accurate mobile localization, automatic driving and so on. Below are two motivating examples about the necessity to detect anomalies over trajectory streams.

EXAMPLE 1. Many senior citizens require constant monitoring and care but such care is expensive without an automatic process. Thus to continuously monitor anomalies of their trajectory streams generated from mobile tracking devices will be very useful, especially when they go outdoor. Such anomalies are very rare patterns that may indicate events such as taking a wrong bus, having a bad fall, encountering a sudden slow-down and getting lost. If their families could be notified in time, immediate and possible life-saving actions can be taken. Figure 1 describes a typical scenario: one day Bob's father takes a strange detour (the red trajectory b) compared to his usual route (the blue trajectory a); then Bob is notified immediately about this abnormal case.

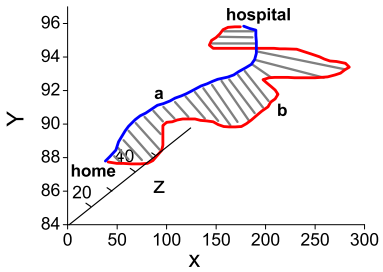


Figure 1: An Elder Monitoring Application

EXAMPLE 2. Nowadays, most road traffic devices like cars, buses and trucks have been installed with GPS systems. Therefore, their trajectory streams can be continuously monitored by a central system. Once there are some emergent accidents, the trajectories of those traffic devices may become abnormal. If those trajectory anomalies are captured by the monitoring system in a short time, some immediate solutions could be conducted to handle the accidents in time.

There are some nice solutions for problems like online event [32, 15, 24], burst [33, 29], and novelty detection [25] over stream time series (or trajectories), where the respective definitions of anomalies are: 1) distribution/model variations from assumed distribution, 2) bursts with statistical aggregations exceeding a threshold, and 3) abnormal classes by classification based on labeled data. Unfortunately, none of the solutions for these anomalies could be extended to find more general anomalies for a great number of applications, because in a trajectory stream, 1) distribution is always changing, 2) burst is only one kind of anomalies, and 3) labeling data has a huge cost. In the 2 examples described above, the anomalies are rare patterns which would have big *spatial deviations (distances)* from the normal trajectories in a certain *temporal interval*, like the red trajectory (trajectory a) in Figure 1.

Thus, in this paper, we would like to focus on continuous monitoring *distance-based anomalies* from trajectory streams, where distance is used as a measure of (dis-)similarity between trajectory subsequences. In fact, the usefulness of distance-based anomalies for general databases has been thoroughly justified in previous work [20] and [21].

Though distance-based anomalies are more general and effective, the performance of anomaly monitoring suffers from intensive

KDD'09, June 28–July 1, 2009, Paris, France.

distance computations, thus efficient pruning strategies become essential for streaming cases. Furthermore, targeting mobile applications, the proposed algorithm should require little extra memory. Naïve solutions like sequential scan will result in excessive time cost, while *R*-tree styled structures will incur huge memory consumptions for indexing all subsequences extracted from a trajectory, concomitant with high update cost. Even with dimensionality reduction, the reduced representation for every subsequence will occupy k_r times more space than the original stream data, where k_r is the dimensionality after reduction. Therefore it is a challenging problem to tame the computation and memory costs of distance-based anomaly monitoring on trajectory streams.

In this paper, we make the following contributions:

- *Efficient Anomaly Monitoring by Local Clusters*: we build local clusters for trajectories in a streaming manner, then anomalies are detected by an efficient cluster join mechanism using pruning strategies without introducing false dismissals.
- *Piecewise Index for Rescheduling Cluster Join*: to further improve the performance, we design a piecewise VP-tree (vantage point tree [10]) based index structure and reschedule the order of cluster joins.
- *Experimental Study*: we test the proposed techniques on real world as well as synthetic data. The results show that our anomaly definition is much more useful than other known definitions and our techniques can achieve orders of magnitude improvement in performance compared to a simple pruning approach.

The rest of the paper is organized as follows. Related work is discussed in Section 2. Section 3 states the problem. Section 4 introduces a preliminary method and Section 5 introduces our pruning methods by local clustering. Section 6 is about the piecewise VP-tree based indexing technique. We present extensive experimental evaluations in Section 7. We conclude in Section 8.

2. RELATED WORK

In this section, we briefly review the related work in two relevant areas: time series data management and anomaly detection.

Since trajectory is a special type of time series, we review some work in time series databases. Agrawal et al.'s pioneering work [1] uses DFT (discrete fourier transform) to reduce the dimensionality of time series and then conducts the search in the reduced dimensional space. Later on, Faloutsos et al. [11] propose a general frame work for similarity search over time series data, called GEMINI framework, to support subsequence matching. The main idea is conducting the search in a filter and refine manner via lower bounds derived from dimension reduction techniques. Subsequent work includes various methods of dimension reduction for time series, such as SVD [22], DWT [7], APCA [18], and Chebyshev Polynomials [6], all of which guarantee no false dismissals. Meanwhile, many novel and effective distance measures for (dis-)similarity between time series are proposed along with corresponding lower bounds, such as DTW (dynamic time warping) [30], LCSS (longest common subsequence) [28], ERP (edit distance with real penalty) [8], EDR (edit distance on real sequence) [9] and probabilistic measure [31]. However, these techniques are designed for similarity search. For anomaly detections in archive time series, Keogh et al. [19] use SAX (symbol approximate aggregation) and heuristic reorder on candidates to find discords. In their work, discords are defined as sequences that have the furthest distances to their nearest neighbors. A Trie structure is used to reduce the search time. However, in our new problem setting, the trie technique is no longer applicable due to the changing feature of data and the high update cost.

With emerging requirements on continuous stream time series management, Zhu and Shasha [33] find statistics over a single stream and correlations among multiple data streams, where incremental DFT is used to prune uncorrelated stream pairs. Vlachos et al. [29] identify bursts based on the computation of the moving average (MA) and propose a novel burst similarity measure by MA and intersections. Gao et al. [12] reduce I/O cost for continuous similarity queries by pre-loading the predicted index pages and archived time series into the allocated cache memory. Bulut et al. [5] provide a well designed multi-resolution hierarchical index for monitoring aggregate and similarity queries over a stream time series. However, none of the above work address the problem of monitoring general anomalies along a stream time series or trajectory streams where no fixed pattern exists.

A lot of research work has been conducted for mining distance-based anomaly (outlier) in traditional databases [14]. The common solution for distance-based anomaly detection applies a nested loop to count range neighbors for every anomaly candidate. Knorr and Ng [20] develop a CELL-based method which can efficiently locate anomalies in very large datasets, yet it is known that the CELL-based method is not scalable to high-dimensional datasets like time series. Ramaswamy et al. [26] rank top n anomalies by distances($D^k(\cdot)$) to their k-th nearest neighbors, where a R^*-tree is used for the k-nearest neighbors search of every point p and a partition-based algorithm is proposed. However, the R^*-tree method cannot be adapted to trajectory streams because it will suffer from continuous high speed update and huge memory cost. Bay et al. [3] search anomalies by a range neighbor search with a random order, together with a simple pruning step, however, it is still quite slow for trajectory streams since too many subsequences need to be checked. Tao et al. [27] prove an upper bound for the memory consumption, which permits the discovery of all anomalies by scanning the dataset three times, but it is not applicable to streams where available data keep changing. Breunig et al. [4] propose the concept of "local outliers", according to which an object is an outlier if it is significantly different from its spatially nearby objects. Jin et al. [16] propose efficient pruning strategies to find "top-k local outliers" quickly. Different from "local outliers", the anomalies we monitor are those trajectory subsequences significantly different from their spatial nearby ones in a certain temporal window, which fit the feature of stream data.

To summarize, traditional anomaly definitions require the access to a global and static database, which could not exist in stream scenarios. In stream scenarios, data keep updating, but the limited memory can only store a limited "recent" sliding window. Of course the solutions for static database anomalies could be applied here, but from the experimental results, we could find that our solution considering the specific features of trajectory streams has got orders of magnitude performance improvement to previous ones.

3. PROBLEM STATEMENT

Before we give the formal definition of the problem, we introduce several terms used in this paper. For simplicity of illustration, trajectories used in the following definitions are 1-dimensional, yet we will see how they can be trivially extended for multi-dimensional trajectories at the end of this section. We define a *trajectory stream* as an totally ordered infinite sequences S =$\{s_1, s_2, ..., s_t, ...\}$, where s_i is a real value arriving at a specific time tick i, and time tick i is after time tick $i-1$. Without loss of generality, in our diagrams, the stream flow always comes from the right side.

Given a trajectory stream S, any subsequence $\mathcal{B}$ =$\{s_i, s_{i+1}, ..., s_{i+w_b-1}\}$ of S, is called a *base window*, where w_b is a predefined base window length. Given a base window $\mathcal{B}$ =$\{s_i, s_{i+1}, ..., s_{i+w_b-1}\}$ of trajectory stream S =$\{s_1, s_2, ..., s_t, ...\}$, the subsequence L =$\{s_{i-w_l}, s_{i-w_l+1}, ..., s_{i-1}\}$ of length w_l is defined as $\mathcal{B}$'s *left sliding window*, the subsequence R =$\{s_{i+w_b}, s_{i+w_b+1}, ..., s_{i+w_r-1}\}$ of length w_r is defined as $\mathcal{B}$'s *right sliding window*,

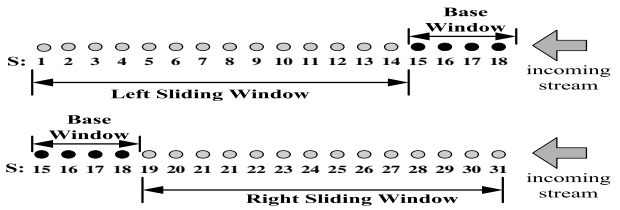


Figure 2: Illustration of the Trajectory Stream Model

where w_l and w_r are pre-defined sliding window lengths. Figure 2 gives a visualized example about the basic constructs of trajectory stream, base window, left sliding window and right sliding window.

DEFINITION 1 (DISTANCE). **Distance(q, c)** *is a function that has base window q and c as inputs and returns a nonnegative value d, which is said to be the distance from q to c.*

Without loss of generality, we take Euclidean distance as the distance measure in our implementations: for base window X and Y, distance(X, Y)=$\sqrt{\sum_{i=1}^{w_b}(X_i - Y_i)^2}$.

DEFINITION 2 (NEIGHBOR). *Given a base window $\mathcal{B}$ ={s_j, s_{j+1},...,s_{j+w_b-1}} which is a candidate currently to be judged whether it is an anomaly, a trajectory stream $\mathcal{S}$ ={s_1, s_2,... ,s_h} where $h \geq w_b$, and distance threshold d, for any subsequence $\mathcal{S}_i$ ={s_i, s_{i+1},..., s_{i+w_b-1}} of $\mathcal{S}$, if $distance(\mathcal{B}, \mathcal{S}_i) < d$, we say that $\mathcal{S}_i$ is a* **neighbor** *of $\mathcal{B}$ in $\mathcal{S}$.*

DEFINITION 3 (TRAJECTORY STREAM ANOMALY). *For a base window $\mathcal{B}$ in S, we can get the number (n_1) of $\mathcal{B}'s$ neighbors in its left sliding window L, and the number (n_2) of $\mathcal{B}'s$ neighbors in its right sliding window R, then $\mathcal{B}$ is a* **trajectory stream anomaly** *if $n_1 + n_2 < k$.*

The basic problem is to determine if a base window $\mathcal{B}$ has less than k neighbors in its left and right sliding windows, and if so, $\mathcal{B}$ is an anomaly. The rationale behind Definition 3 is the observation that in real world applications, anomalies in a trajectory stream usually not only have salient spatial deviations from both preceding and succeeding normal subsequences, but also last for a short period. In fact, other than the sliding window, the anomaly definition is the same as the distance-based outliers defined in [20]. In other words, we adopt traditional distance-based anomaly in stream scenarios. However, the solution proposed for detecting distance-based anomalies in traditional static databases is not applicable to trajectory streams due to high-speed data update of stream data. Now we give an introductive definition of the anomaly monitoring problem.

DEFINITION 4 (PROBLEM). *On trajectory stream S, at every time tick t, upon the arrival of s_t, we check whether the base window $\mathcal{B}$ ending at s_{t-w_r} is a trajectory stream anomaly.*

In practice, we recommend applications to take a vary large left sliding window length w_l and a short right slide window length w_r to ensure quick responses to anomalies, because for real anomaly cases, part of the monitoring procedure for the base windows is deferred to the time when its whole right sliding window comes up. Note that although our problem setting is to monitor anomalies over a stream from a single data source, we can simply extend the definition to handle multiple streams from not only homogenous but also heterogeneous data sources. For homogenous streams, for every new coming base window on one trajectory stream, we could accumulate the count of its neighbors on all other trajectory streams. For heterogeneous streams, we can merge streams from different sources into one multi-dimensional stream. Then, to extend our problem definition for a m-D trajectory stream, we only need to extend the distance measure: for base window X and Y, distance(X, Y)=$\sqrt{\sum_{i=1}^{w_b}\sum_{j=1}^{m}(X_i[j] - Y_i[j])^2}$.

4. A PRELIMINARY SOLUTION: SIMPLE PRUNING

Obviously, directly computing the anomalies tick by tick is computationally expensive. For each base window $\mathcal{B}$ in the trajectory stream, we need n_1 and n_2, the numbers of neighbors in the left and right sliding windows of $\mathcal{B}$, respectively. If $n_1 + n_2 < k$ then $\mathcal{B}$ is an anomaly. One can adopt a more efficient method by random sampling suggested by [3] for outlier detection in traditional databases. To gather the count of neighbors one can repeatedly pick some base window $\mathcal{B}'$ in the left or right sliding window of $\mathcal{B}$ in a random manner, and check whether $\mathcal{B}'$ is $\mathcal{B}$'s neighbor. If $\mathcal{B}$'s neighbor count reaches k, $\mathcal{B}$ is certified to be non-anomaly and remaining search for $\mathcal{B}$ can be simply pruned. We call this random sampling method *simple pruning*. Simple pruning can be seen as a set of independent $Bernoulli$ trials where we keep drawing samples until k successes or the whole dataset is exhausted. The following theorem [3] gives an analytic result of time cost by simple pruning.

THEOREM 1 (COST OF SIMPLE PRUNING). *Let $F_x(d)$ be the probability that a randomly drawn sample lies within distance d to base window x, P_a be the anomaly rate, $N = w_l + w_r$, let Y be a random variable representing the number of trials (distance computations) until we have k successes, and let $P(Y = y)$ be the probability of obtaining the k-th success on trial y. Then the expectation of Y follows:*

$$E(Y) \leq \frac{k}{F_x(d)} + P_a N \tag{1}$$

Since P_a is tiny, $E(Y)$ is dominated by $\frac{k}{F_x(d)}$, which is independent of the sliding window size. In most trajectory stream applications, data arrive at a very high speed, thus it is still of great challenges for us to design exact and more efficient algorithms beyond simple pruning, in order to avoid data buffer overflow.

5. EFFICIENT MONITORING BY LOCAL CLUSTERS

Our goal is to do better than the simple pruning technique. Our idea is based on the observation that most trajectory streams are locally continuous, so that two highly overlapping base windows tend to have a short distance. We propose a local clustering-based approach to monitor anomalies, which utilizes the property of local continuity. In fact, many current time series/trajectory indexing methods have implicitly assumed the underline time series data have the local continuity property. It is known that neither DFT [1], DWT [7], APCA [18] for Euclidean distance, nor LB_Keogh [17], LB_Zhu [34] for DTW distance could get a satisfactory pruning power on a highly fluctuant time series or trajectories.

In Section 5.1, we introduce an online local clustering algorithm. Then, batch monitoring is established with pruning strategies in Section 5.2. Finally we give a detailed cost analysis in Section 5.3.

5.1 Incremental Local Clustering

In the following we define local cluster $\mathcal{C}$ as a trajectory subsequence of length at most m_b in which there exists a base window $\mathcal{B}$ that all other base windows in $\mathcal{C}$ are within a certain distance τ from $\mathcal{B}$. One such base window $\mathcal{B}$ is chosen as the *pivot* of $\mathcal{C}$. Actually local clusters are a special kind of temporal partitions on a trajectory stream, and each cluster contains a number of consecutive data points within a stream. We shall show that local clusters are easy to build and can be incrementally updated, which fits the requirement of continuous trajectory streams perfectly. How to determine m_b and τ will be discussed later.

Algorithm Online Local Clustering
Input: latest base window B_{new}, latest time tick t
Global variables: distance threshold τ, temporal constraint m_b, boolean $pivotfound$, current pivot $p_{current}$, current radius r, current left bin $\mathcal{L}$. (Initially $pivotfound \leftarrow$ FALSE; $p_{current} \leftarrow$NULL, $r \leftarrow 0$; $\mathcal{L} \leftarrow \phi$.)

1: **if** $pivotfound$ = FALSE **then**
2: *(Comments: accumulate points in the left bin)*
3: **if** the distance $dist_i$ between B_{new} and any base window B_i ending in the current left bin $> \tau$ **or** current cluster size = m_b **then**
4: the current left bin minus time tick $t-1$ is the final left bin
5: set time tick $t-1$ as the pivot $p_{current}$
6: $pivotfound \leftarrow$ TRUE
7: **else**
8: time tick t is added to the current left bin $\mathcal{L}$
9: update r to be the greatest $dist_i$
10: **end if**
11: **end if**
12: **if** $pivotfound$ **then**
13: *(Comments: accumulate points in the right bin)*
14: find distance $dist$ between B_{new} and the base window ending at $p_{current}$
15: **if** $dist > \tau$ **or** current cluster size is m_b **then**
16: time ticks in $\{p_{current}, ..., t-1\} \cup \mathcal{L}$ forms a local cluster
17: set current left bin $\mathcal{L}$ to $\{t\}$, $r \leftarrow 0$,
18: $p_{current} \leftarrow$ NULL, $pivotfound \leftarrow$ FALSE
19: **else**
20: **if** $dist > r$ **then**
21: $r \leftarrow dist$
22: **end if**
23: **end if**
24: **end if**

Figure 3: Online Local Clustering Procedure

DEFINITION 5 (LOCAL CLUSTER). *With a distance threshold τ and a temporal constraint m_b, given a sequence $\mathcal{C} = \{s_i, s_{i+1}, ..., s_j\}$, $j - i + 1 \leq m_b$, which is a piece of a trajectory stream S, if $\exists s_k \in \mathcal{C}$, $\forall s_x \in \mathcal{C}$, $\mathcal{B}_x$ and $\mathcal{B}_k$ being base windows ending at s_x and s_k respectively, $distance(\mathcal{B}_k, \mathcal{B}_x) \leq \tau$, $\mathcal{C}$ is called a* **local cluster**.

DEFINITION 6 (PIVOT, RADIUS, LEFT BIN AND RIGHT BIN). *For a local cluster $\mathcal{C}$ of distance threshold τ, a special point $s_x \in \mathcal{C}$ is selected as $\mathcal{C}$'s* **pivot**, *s_x satisfies the condition that $\forall s_i \in \mathcal{C}$, $\mathcal{B}_x$ and $\mathcal{B}_i$ being base windows ending at s_x and s_i respectively, distance($\mathcal{B}_x$, $\mathcal{B}_i$)$\leq \tau$. The value of maximum$_{s \in \mathcal{C}} distance(s_x, s)$ is called $\mathcal{C}$'s* **radius**. *Points whose time ticks are less (greater) than x in $\mathcal{C}$ constitute pivot s_x's* **left bin** *(***right bin***).*

The concepts of pivot, radius, left bin and right bin are used in the algorithm of online local cluster construction. The parameter m_b specifies that at most how many based windows could be included in a local cluster, while the parameter τ specifies the upper bound of distances from the pivot to all base windows in the local cluster. Both parameters need to be specified for local clustering.

In the data structure of a local cluster, 4 variables are stored: its pivot's position on the sliding window, the start and end positions that it covers, and its radius. In the following part, we use pivot to denote either the base window ending at its position or the corresponding local cluster data structure. Figure 3 shows the online local clustering algorithm, where a greedy approach is used to form local cluster partitions. In this algorithm, for a resulting cluster $\mathcal{C}$, each time tick t_i in the left bin requires at most l_i distance computations if there are l_i elements aligned on t_i's left but within $\mathcal{C}$. For time ticks in the right bin, each requires a single distance computation. Thus if the average cluster size is m then the average number of distance computations per time tick is at most $m/2$.

5.2 Batch Monitoring by Cluster Join

With local clusters, we search for neighbors in a batch manner. Neighbor search for base window $\mathcal{B}$ is firstly triggered upon the formation of $\mathcal{B}$'s local cluster, and secondly called upon the arrival of the rightmost point in $\mathcal{B}$'s right sliding window if $\mathcal{B}$'s accumulated neighbor count has not reached k. In order to achieve the neighbor search we must keep all needed data points, which we call the current sliding window W. Let t_{o1} be the oldest time tick in the local clusters that overlaps with the left sliding windows of the new data points that have not been clustered, and t_{o2} be the oldest time tick of a local cluster whose entire right sliding window has not yet arrived. W is made up of all time ticks from $min\{t_{o1}, t_{o2}\}$ to the current time tick.

There are two concurrent steps in the batch monitoring:

- When a new local cluster $\mathcal{C}_{new}$ is constructed, we join it with the preceding clusters in a random order (by a *clusterjoin* function) in W until every base window in $\mathcal{C}_{new}$ has more than k neighbors. If we exhaust all possible candidate base windows in the left sliding window and cannot find k neighbors for base window $\mathcal{B}$ in $\mathcal{C}_{new}$, we keep the neighbor count for $\mathcal{B}$.
- When the entire right sliding window of all base windows in local cluster $\mathcal{C}_{old}$ are arriving, in $\mathcal{C}_{old}$, if any base window $\mathcal{B}$'s neighbor count is kept in step (1), $\mathcal{B}$'s neighbor count in its right sliding windows are determined in a similar way. The neighbor count n_1 collected in step (1) for a base window $\mathcal{B}$ is to be added to the count n_2 in this step to give the total neighbor count.

In the two batch monitoring steps, the *clusterjoin* function is to accumulate the neighbor count in some cluster $\mathcal{C}$ for each base window in $\mathcal{C}_{new}$, which is a critical and expensive operation. Note that if the average cluster size is m, the brute force join cost is $O(m^2)$. However, based on properties of the local clusters, we could utilize several rules to enhance efficiency as follows.

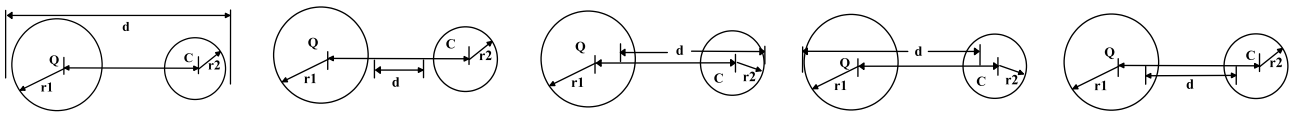
LEMMA 1. *Given two local clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, if $distance(\mathcal{C}_1.pivot, \mathcal{C}_2.pivot) + \mathcal{C}_1.radius + \mathcal{C}_2.radius < d$, then $\forall$ base window pair $\mathcal{B}_i$ and $\mathcal{B}_j$, $\mathcal{B}_i \in C_1$ and $\mathcal{B}_j \in \mathcal{C}_2$, $distance(\mathcal{B}_i, \mathcal{B}_j) < d$.*

LEMMA 2. *Given two local clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, if $distance(\mathcal{C}_1.pivot, \mathcal{C}_2.pivot) - \mathcal{C}_1.radius - \mathcal{C}_2.radius > d$, then $\forall$ base window pair $\mathcal{B}_i$ and $\mathcal{B}_j$, $\mathcal{B}_i \in \mathcal{C}_1$ and $\mathcal{B}_j \in \mathcal{C}_2$, $distance(\mathcal{B}_i, \mathcal{B}_j) > d$.*

LEMMA 3. *Given a base window $\mathcal{B}$ and a local cluster $\mathcal{C}$, if $distance(\mathcal{B}, \mathcal{C}.pivot) + \mathcal{C}.radius < d$, then $\forall$ base window $\mathcal{B}_i \in \mathcal{C}$, $distance(\mathcal{B}, \mathcal{B}_i) < d$.*

LEMMA 4. *Given two base windows $\mathcal{B}_i \in \mathcal{C}_1$ and $\mathcal{B}_j \in \mathcal{C}_2$, consider base windows as points and local clusters as hyper-balls in high-dimensional space with their pivots as centroids, the perpendicular bisector hyper-plane $\mathcal{P}$ of $\mathcal{C}_1$ and $\mathcal{C}_2$, if $distance(\mathcal{C}_1.pivot, \mathcal{C}_2.pivot) - \mathcal{C}_1.radius - \mathcal{C}_2.radius < d$ and $distance(\mathcal{B}_i, \mathcal{P}) + distance(\mathcal{B}_j, \mathcal{P}) > d$, then $distance(\mathcal{B}_i, \mathcal{B}_j) > d$.*

The 4 lemmas allow us to devise pruning strategies that guarantee the correctness in anomaly monitoring. Totally, there are 5 cases for the spatial relationship between two local clusters, as depicted in Figure 4(a) to Figure 4(e). In Case 1 all elements in cluster $\mathcal{C}$ are neighbors of all elements in the query cluster $\mathcal{Q}$, and in this case, joining the two clusters requires only one distance computation (between their pivots). In Case 2 no element in cluster $\mathcal{C}$ is the neighbor of any element in query cluster $\mathcal{Q}$, and in this case, to join the two clusters requires only one distance computation, too. In Case 3 where all elements in cluster $\mathcal{C}$ are neighbors of some elements in query cluster $\mathcal{Q}$, and for those elements in $\mathcal{Q}$, only one distance computation is needed. In Case 4 some elements in cluster $\mathcal{C}$ are neighbors of all elements in query cluster $\mathcal{Q}$, and for those elements in $\mathcal{C}$, only one distance computation is needed. In Case 5 some elements in cluster $\mathcal{C}$ are neighbors of some elements in query cluster $\mathcal{Q}$, in this undesirable case, still some distance computations could be pruned by Lemma 4, which is illustrated in Figure 5.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4 (e) Case 5

Figure 4: Local Cluster Relationships

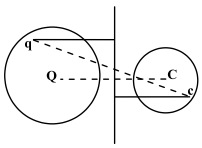


Figure 5: Lemma 4

With the above analysis, we can join clusters by checking which case the relationship belongs to, and then prune the distance computations. Note that situations of Cases 3, 4 and 5 are overlapping, so that some situation may belong to all the 3 cases. Therefore, we apply pruning in the order of Case 3, Case 4, then Case 5.

By our proposed batch monitoring scheme with local clustering and pruning rules, we can guarantee the correctness (in terms of Definition 3) of retrieved result set (no false dismissal). However by other clustering algorithms such as *k*-means, it will be difficult to guarantee this point. The difference is that local clustering requires not only the distances of base windows are close, but also the base windows in it are continuous. No existing solution considers this point. Good clustering is not our objective, but good pruning power in processing continuous anomaly monitoring is.

5.3 Cost Analysis and Optimization

Now we analyze the cost of local clustering based batch monitoring, where both the benefit and overhead of local clustering are considered.

In our analysis, it is assumed that the sliding window is of sufficient length (this models *very large databases*), so we could increase the neighbor counts only when Case 1 in Figure 4 is encountered, and ignore the other 4 cases. Although this may not be optimal, it is enough to theoretically show our superiority over simple pruning.

Let m be the average cluster size. We can think of the process of joining a query cluster with candidate local clusters as independent $Bernoulli$ trials where we keep drawing candidates until $\lceil k/m \rceil$ successes appear or all candidates are exhausted. Hence, the number of trials follows the *Pascal* distribution. Assuming the average radius of clusters is r, we have the following analytic results: Theorem 2 gives the time cost per time tick, while Corollary 2 proposes the optimized setting for cluster size.

THEOREM 2 (TIME COST OF BATCH MONITORING). *Let $F_x(d)$ be the probability that a randomly drawn example pivot lies within distance d to query cluster's pivot x, and P_a is the rate of anomalies in the data set at time t. Value s_t at time tick t is within the left bin or right bin of x, and N= $w_l + w_r$. Then the expected number of distance computations Y at the time tick t is upper bounded by:*

$$E(Y) \leq \frac{k}{m^2 F_x(d-2r)} + P_a N + \frac{m}{2} \tag{2}$$

PROOF. The total expected cost of batch monitoring consists of 3 parts: the cost expectation on neighbor search for normal cases, the expected cost on search for clusters with anomalies, and the cost of building local clusters. We first analyze the expected distance computations Z during cluster joins for false candidate cluster which does not contain any anomaly. The process will definitely stop at some join when the candidate's neighbor count exceeds threshold k. Then, the following formula could be derived:

$$\begin{aligned}
E(Z) &= \frac{1}{m}\sum_{z=\frac{k}{m}}^{\frac{N}{m}} P(Z=z)z \\
&= \frac{1}{m}\sum_{z=\frac{k}{m}}^{\frac{N}{m}} \binom{z-1}{\frac{k}{m}-1} F_x(d-2r)^{\frac{k}{m}}(1-F_x(d-2r))^{z-\frac{k}{m}})z \\
&\leq \frac{1}{m}\sum_{z=\frac{k}{m}}^{\infty} \binom{z-1}{\frac{k}{m}-1} F_x(d-2r)^{\frac{k}{m}}(1-F_x(d-2r))^{z-\frac{k}{m}})z \\
&= \frac{k}{m^2 F_x(d-2r)}
\end{aligned}$$

Due to pruning by case 2, the cost for clusters with anomalies is

$$E(C_{anomaly}) \leq P_a N$$

The cost of local clustering incurring on every data point should be included, which is:

$$E(C) \leq \frac{m}{2}$$

E(Y) = E(Z+$C_{anomaly}$+C)=E(Z)+E($C_{anomaly}$)+E(C), thus Inequality 2 holds. □

COROLLARY 1 (OPTIMAL CLUSTER SIZE). *Ignoring the cost for anomaly cases (which is a small constant for any algorithm), the cost of batch monitoring on every time tick could be optimized when:*

$$m = \sqrt[3]{\frac{4k}{F_x(d-2r)}} \tag{3}$$

then, the upper bound of minimized cost $min(E(Y))$ could be obtained:

$$min(E(Y)) \leq 1.2\sqrt[3]{\frac{k}{F_x(d-2r)}} \tag{4}$$

PROOF. The cost in right side of Inequality 2 could be minimized when its partial derivative on m is 0, thus, we let:

$$\frac{\partial E(Y)}{\partial m} = \frac{1}{2} - \frac{2k}{m^3 F_x(d-2r)} = 0$$

Therefore, we could get:

$$m = \sqrt[3]{\frac{4k}{F_x(d-2r)}}$$

Applying Equation 3 to Inequality 2, Inequality 4 holds. □

Following Corollary 1, we periodically reset m_b= $\sqrt[3]{\frac{4k}{F(d-2r)}}$ and τ=$F^{-1}(m_b F(r))$, where F (distance distribution) and r are aggregated from the past stream. At initialization, we set m_b=6w_b and τ=d/8. Note that the initialization has little impact on the long term performance of continuous monitoring.

Once a cluster is formed, it will never be changed; therefore there is no computational maintenance cost, but only some memory cost to store the cluster data structure (defined in Section 5.1), which is given as follows, measured by memory units (4 bytes a unit).

PROPERTY 1 (MEMORY COST BY BATCH MONITORING). *Since each local cluster structure stores 4 variables, the additional memory cost is* $Cost_{BM} = \frac{4|W|}{m}$, *which is very small, compared to the basic necessary memory* $|W|$. □

REMARK 1 (BATCH MONITORING V.S. SIMPLE PRUNING). *For trajectory streams with local continuity, attested by our experiments,* r *usually is tiny compared to* d. *Hence,* $F_x(d - 2r) \approx F_x(d)$. *Thus the search cost of batch processing is nearly the cost of simple pruning divided by* m^2, *where* m *is the average size of local clusters. With ideal cluster size, the cost could be reduced to nearly* $\sqrt[3]{\text{cost of simple pruning}}$. □

6. INDEXING AND JOIN RESCHEDULING

Through local cluster-based pruning, the distance computations will be largely reduced. However, most distance computations are spent on the normal cases since most base windows extracted from the trajectory streams are not anomalies. Hence, looking at the batch monitoring, if candidate clusters on W are joined in a perfect order that the query cluster first joins with candidate clusters which satisfy the condition of case 1 in Figure 4(a), the query could be proven to contain no anomalies quickly by very few distance computations (only one distance computation for candidate cluster C, while the query's neighbor count increases by $|C|$). If we examine Figure 4 again, case 3 or case 4 is more likely to give rise to more neighbors for the query cluster with less distance computations than case 5. Hence we use

$$\text{utility of } C = \frac{\text{increase in number of neighbors to } Q}{\text{number of distance computations}}$$

to measure the goodness of a candidate cluster. Obviously the greater the utility is, the better the candidate is; and vice versa. Comparing case 3, 4, and 5 in the above, the distance between the pivot of a candidate cluster and pivot of a query cluster usually can indicate the utility: smaller distance corresponds to higher utility.

Therefore, our basic idea is to build index for pivots along current sliding window and then reschedule the cluster join order so as to approach the best search order. We propose a novel way to index trajectory stream by piecewise VP-trees (Vantage Point Tree [10]). The choice of VP-trees is because they occupy less memory (since only pivots need to be maintained in a tree node) compared to *R*-tree's bounding boxes. Actually, any tree index structures that do not store high dimensional vectors in their internal nodes could be employed here.

We build piecewise VP-trees, in particular a different VP-tree is built for each fixed size piece or interval of the trajectory stream. The piecewise indexing method allows us to remove an entire VP-tree on data expiry, which is much more efficient than the use of a single index tree where the expired data need to be deleted one by one.

6.1 Piecewise VP-trees

Figure 6 displays an overview of the structure of piecewise VP-trees over a trajectory stream: each VP-tree is to index pivots of a continuous period, and then pivots the whole sliding window are indexed by v VP-trees, from VP-tree 1 to Vp-tree v, except some boundary ones near the start or end of the sliding window. The basic unit of piecewise VP-trees is shown in Figure 7, where VP-tree i indexes an array of pivots (black points) in the box, which correspond to the local clusters spanning over a piece of a trajectory stream from time ticks $21,000$ to $32,000$. Next we introduce how the piecewise index structure works and consider the *insert*, *delete*, and *query* operations on the piecewise index.

Tree Insertion. When a new data point appends to the trajectory stream, the online local clustering procedure is triggered. When a new local cluster with its pivot P_{new} is formed, it is immediately

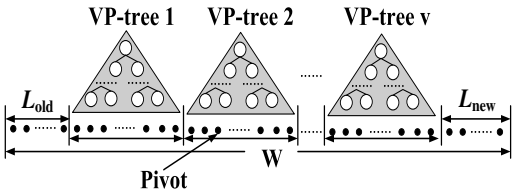


Figure 6: Piecewise VP-trees along the Sliding Window

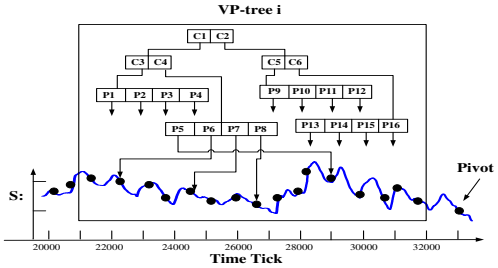


Figure 7: A VP-tree Unit on the Sliding Window

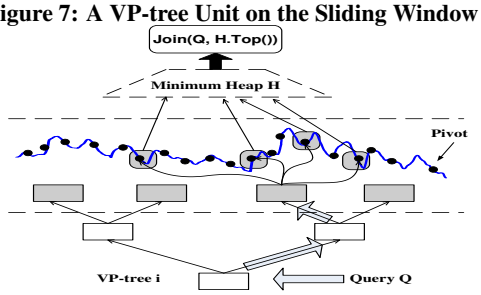


Figure 8: Join Reschedule

put into a list L_{new} (as shown in Figure 6) for new coming pivots. The list L_{new} is continuously updated until its size reaches the predefined number (T_N) of pivots indexed by one VP-tree, at which point a new VP-tree is built from the pivots in L_{new} and inserted into the VP-tree list L_{VP}, and L_{new} is reset to empty. Note that once a VP-tree is built, the only function of it is to answer range queries targeting at the pivots which are indexed by it, and will never be updated all its life time.

Tree Deletion. When some data point in the scope of a VP-tree phases out, the VP-tree is removed from L_{VP}, all its pivots will be inserted into a list L_{old} (as shown in Figure 6). Once a data point s_t phases out, the "start position" of its pivot P_{old} (also current oldest pivot) will be changed to $t + 1$. When P_{old}'s end position equals to its start position, P_{old} will be removed from L_{old}.

Query Processing. When a new local cluster C_{new} is formed, we can accumulate its neighbor counts in the left sliding windows with the assistance of VP-trees. Also when the entire right sliding window of local cluster C_{old} have arrived, we can count the total neighbors of base windows in C_{old} to uncover anomalies. In either case, C_{new} or C_{old} becomes a query cluster Q. Given a query cluster Q with radius r_Q, let r_{max} be the maximum radius of the current set of local clusters, for anomaly detection, a special case of range query with $(d + r_{max} + r_Q)$ (see Definition 2 for d) as the range is issued. When the number of neighbors of every base windows in the query cluster reaches k (see Definition 2 for k), the query processing is stopped. This is called the *stopping condition* for Q ($\forall q \in Q$, q's neighbor count is larger than k).

6.2 Join Rescheduling

When a batch search (range query) is issued, we first look for the query cluster's spatially nearby clusters through the VP-tree list. However, the trick here is that we do not join the query cluster with

selected candidate clusters immediately when they are found, for the reason that in worst case, the join cost for two local cluster pair of size m will require m^2 distance computations. Instead, we first enter candidate local clusters' pivots into a candidate list, which is to be used by the upcoming query optimization steps.

When a new local cluster C_{new} is constructed (or the whole right sliding window of C_{new} is arriving), we search its spatially nearby local clusters in the VP-trees one by one until the stopping condition is reached or all trees are searched. Once the candidate local cluster list is obtained after executing the query on one VP-tree, candidate clusters in the list are put into a minimum heap, where the ranking of clusters is based on the distances between candidate local clusters' pivots and the query cluster's pivot. The smaller is the distance, the higher layer the candidate is placed in the minimum heap. The original batch monitoring procedure is now reordered. Figure 8 demonstrates an example about how the rescheduling works on one VP-tree. For the query Q, we first get 4 candidate pivots through the range search in VP-tree i, then the 4 candidates are put into the minimum heap H, the ranking of which is based on distance between candidates' pivots and Q's pivot. Finally, Q keeps joining with the top element of H until the stopping condition is met or all candidates are searched.

6.3 Cost Analysis

The above strategy improves efficiency from both VP-trees' pruning power and join rescheduling. Even without rescheduling, assuming the average pruning power of the VP-trees is $p\,(0 \leq p < 1)$, the search cost could be reduced to $(1-p) \cdot E(Y)$ (see E(Y) in Theorem 2). With rescheduling, the data processing speed will be even faster. Property 2 and 3 analyze the extra cost during index construction and extra requirements for memory.

PROPERTY 2 (TREE CONSTRUCTION COST). *Given that n is the total number of pivots along current sliding window, n_{vp} is the number of pivots indexed by one VP-tree, n_l is the number of pivots indexed in one leaf node, m is average cluster size, the number of distance computations during tree construction incurred for every time tick is:*

$$Cost_{build} = \frac{nlog_2\lceil\frac{n_{vp}}{n_l}\rceil}{nm} = \frac{1}{m}log_2\lceil\frac{n_{vp}}{n_l}\rceil \qquad (5)$$

Obviously, in most cases, compared to the cost $E(Y)$ (in Theorem 2), $Cost_{build}$ is small. □

PROPERTY 3 (MEMORY COST OF PIECEWISE VP-TREES). *Given m and n_l as in Property 2 ,$|W|$ (the current sliding window length), we derive the extra memory cost for VP-trees:*

$$Cost_{VP} = n_l\frac{|W|}{mn_l} + 4(\frac{|W|}{mn_l} - 1) \approx \frac{(n_l+4)|W|}{mn_l} \qquad (6)$$

$Cost_{VP} \approx \frac{1}{4}Cost_{BM}$ *($Cost_{BM}$ is in Property 1), since normally n_l is much larger than 4.* □

The tree size n_{vp} and leaf size n_l could be pre-tuned for different applications.

7. EXPERIMENTAL EVALUATIONS

In this section, we evaluate both the effectiveness and efficiency of our anomaly monitoring framework by extensive experimental studies. Both real world datasets (movement, spaceshuttle, ge, altitude, latitude, longitude) and synthetic dataset (random-walk)[1] are used. A Pentium IV 2.2GHz PC with 2GB RAM is used to conduct all our experiments. In our implementation, we simulate the streaming manner by using a sliding window in the main memory. First we load a trajectory dataset into the main memory, and initialize the sliding window's start and end position. Then we keep sliding the sliding window. When the sliding window slides one point forward, the anomaly detection procedure is triggered, and when the procedure is completed, the sliding window moves one point forward again. For temporally overlapping anomalies, we only report the first discovered one which can represent the whole problematic region.

[1]All the datasets and descriptions are available at: http://www.cse.ust.hk/~leichen/sigkdd09-repository/index.htm

7.1 Effectiveness

We test the usefulness of trajectory stream anomaly (as Definition 3) on 2 annotated datasets "movement" and "spaceshuttle". "Spaceshuttle" contains a series of space shuttle marotta values. "Movement" (MV) contains 12 days' 3D trajectories (35170 data points and 3 dimensions for each point) of one person on his way between home and office, and is annotated to indicate where are the really meaningful anomaly subsequences, including special events and localization errors. We measure the quality of reported anomaly base windows by *F-measure* [2] as follows:

$$\textit{F-measure} = \frac{2 \cdot \textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

$$\textit{Precision} = \frac{|R_o \bigcap D_o|}{|R_o|} \qquad \textit{Recall} = \frac{|D_o \bigcap R_o|}{|D_o|}$$

where D_o is the set of annotated anomalies in a dataset, R_o is the set of "anomaly" results reported by an algorithm (perhaps including false alarms). Here X$\bigcap$ Y=$\{w_i | w_i \in$ X, $\exists\ w_j \in$Y, w_i overlaps with $w_j\}$. Note that we will explain detailed results based on "movement" in Figure 9, while similar results on "spaceshuttle" could be derived from Figure 10.

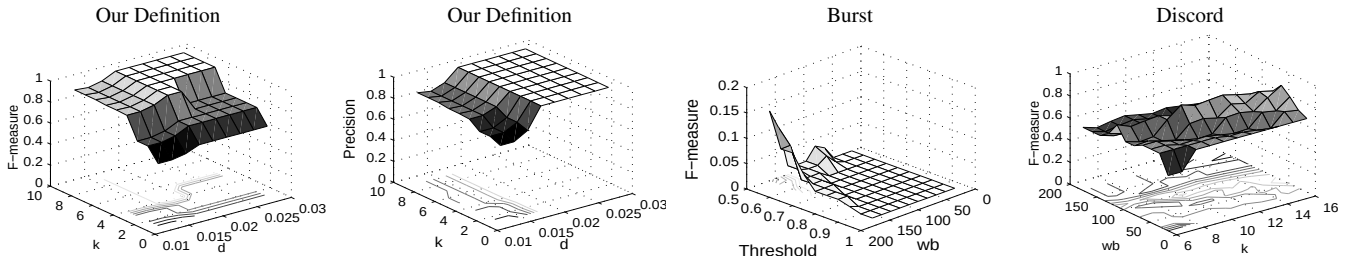
7.1.1 Comparing with Other Definitions

We compare our definition of trajectory stream anomaly with 2 counterparts: *burst* [33] and *discord* [19], both of which could capture a portion of meaningful anomalies. Burst is a period on trajectory streams with aggregated sum (for m-D base window X, aggr(X)=$\sum_{j=1}^{m} |\sum_{i=1}^{w_b} X_i|$) exceeding a threshold, while top-k discords are the subsequences with the top-k largest distances to its nearest neighbor in archived time series. In the experiment for bursts, we vary the threshold (the metric is its proportion over the largest aggregated sum) and the base window length w_b to investigate how F-measure is impacted, and the results are shown in Figure 9(c). Similarly, by treating the dataset as an archived one, we search top-k discords under different base window length w_b, and Figure 9(d) shows F-measure of reported top-k discords for different (k, w_b). From the results, we find that F-measure is always less than 0.15 for burst, and less than 0.82 for discord no matter how their parameters are tuned. From Figure 9(a) once the parameters for our definition fall in a rather loose range, the F-measures is nearly 100%. Figure 9(b) shows that the precision of our definition is also good and robust. Besides results from "movement" dataset, similar results have been observed on "spaceshuttle" dataset, which confirm that our definition is superior. The rationale is that burst misses spatial deviations and discord misses similar anomalies, but our definition covers them.

7.1.2 Parameter Setting

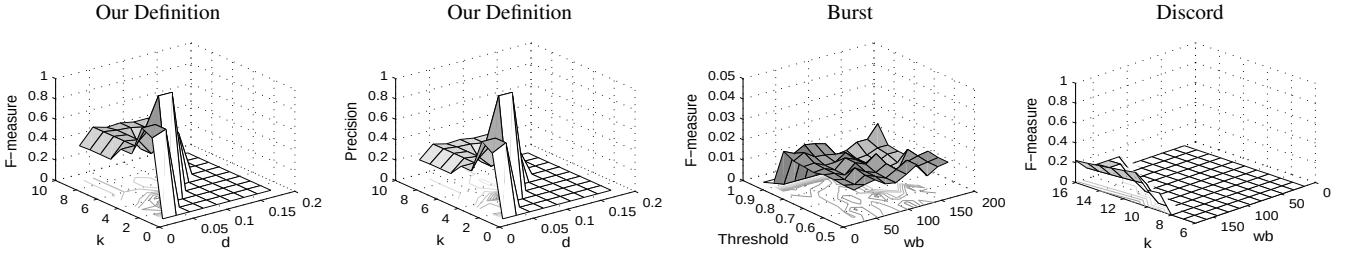
We investigate the impact of parameter setting on "movement" dataset.

Effect of k and d. We vary k and d while let w_b=64 and sliding window be the whole dataset to investigate the effectiveness of our definition trajectory stream anomaly, the results of which are demonstrated in Figure 9(a). Here the metric of d is its proportion over the



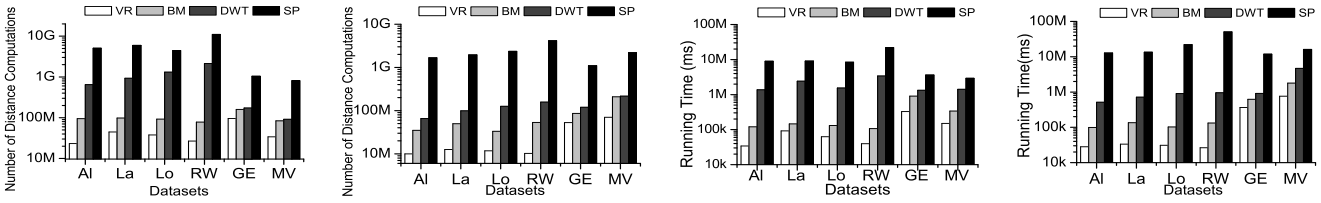
(a) F-measure vs. (k,d) (b) Precision vs. (k,d) (c) F-measure vs. (Threshold, w_b) (d) F-measure vs. (k, w_b)

Figure 9: Effectiveness Comparisons With Burst and Discords("movement")



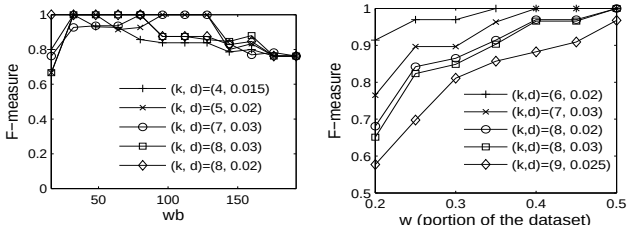
(a) F-measure vs. (k,d) (b) Precision vs. (k,d) (c) F-measure vs. (Threshold, w_b) (d) F-measure vs. (k, w_b)

Figure 10: Effectiveness Comparisons With Burst and Discords("spaceshuttle")



(a) Pruning Power, w_b=128 (b) Pruning Power, w_b=256 (c) Running Time, w_b=128 (d) Running Time, w_b=256

Figure 12: Efficiency vs. Datasets



(a) F-measure vs. w_b (b) F-measure vs. $|W|$

Figure 11: Parameter Effects

diameter of the dataset (the farthest distance between a base window pair). It could be found that when k and d lie in a rather loose range ($0.015 \leq d \leq 0.03$, $4 \leq k \leq 10$), the F-measure is 100%. Moreover, the precision is 100% when k and d vary in a even larger range, as in Figure 9(b).

Effect of Base Window Length w_b. Figure 11(a) shows how F-measure changes with w_b under different (k, d) configurations. Of course w_b depends on the applications and the length of periods that people are interested in. Yet here we find that our anomaly detection results usually are not very sensitive to w_b.

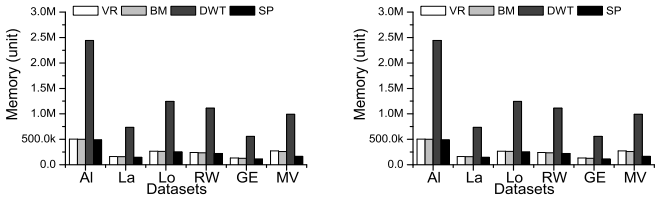
Effect of Sliding Window Length $|W|$. In this experiment, we change the current sliding window length $|W|$, while let w_b=64, to see how the F-measure is affected by $|W|$. From the results in Figure 11(b), we can conclude that no matter how (k, d) is configured, the larger $|W|$ is, the better the monitoring quality will be. Therefore, applications should set the sliding window as long as possible.

Suitable Range for All Parameters. From more experiments, we find that when k is $4 \sim 10$, d is 0.015 to 0.03 of diameter, and w_b is $32 \sim 256$, the F-measures are always high. For $|W|$, it should better be as large as the memory could hold.

7.2 Efficiency

Since our solutions aim at high speed and large scale moving object trajectory streams, which differ from other work where usually small or non-streaming datasets are used, there are 6 very large trajectory datasets involved in the efficiency experiments: altitude (Al), latitude (La), longitude (Lo), random-walk (RW), ge (GE), and movement (MV).

We compare the costs of the 4 algorithms: *SP* (simple pruning) [3], *DWT* (discrete wavelet transform) [7], *BM* (our batch monitoring) and *VR* (our VP-tree based rescheduling), by the number of distance computations they called, total running time and memory consumptions. Similarity search techniques like SVD [22], DWT [7], APCA [18], and Chebyshev Polynomials [6] could naturally be employed for our monitoring procedure. Since DWT has similar performance with those peers and is widely used, we choose DWT [7] as a representative to compare with our algorithms, where truncated wavelet coefficients for each candidate subsequence are maintained in order to obtain distance lower bounds. The number of distance computations of DWT consists of the number of both non-pruned distance computations on subsequences and conversions of computations for lower bound distances on coefficients. In these experiments, parameters are set as: w_b=128 and 256. We vary k from 8 to 15, change d from 0.015 to 0.025 diameter, and record an average performance for the 4 competitive algorithms. Sliding window length $|W|$ for each dataset is as follows: 244159 (Al), 73410 (La), 124442 (Lo), 111139 (RW), 55832 (GE), and



(a) w_b=128 (b) w_b=256

Figure 13: Consumed Memory vs. Datasets

161122 (MV), and we slide forward the sliding window 650000 ticks for each dataset (loop sliding if the end of dataset is touched). The reduced dimensionality of DWT coefficients is 8 (for MV, it is 2 because larger dimensionality will lead to heap memory overflow!). We use very large sliding windows in order to simulate real applications' workloads.

From the results in Figure 12(a), (b), (c) and (d), we could see clearly that VR gets orders of magnitude improvement to SP (on average 179.87 times speedup) and beats DWT significantly (on average 35.8 times speedup). Thus, even if DWT could only use the first coefficient to achieve the same pruning power, VR is still much more efficient. With VR, the average processing time per data point arrival is $0.05 \sim 0.5$ milliseconds, while with SP it is $13.7 \sim 33.6$ milliseconds! From Figure 13(a) and (b), we find that VR consumes little extra memory while DWT requires much more memory to store coefficients for each candidate subsequence. We also compare the 4 algorithms under other parameter settings, and get similar results. Since all the 4 algorithms guarantee the correctness and get the same results, the efficiency improvement by our techniques has no quality regression, and consumes little extra memory. The weakness of VR is the same as most other time series or trajectory indexing work. In experiments, it could not get significant performance growth on highly fluctuant stream time series.

8. CONCLUSIONS

In this paper, targeting typical scenarios of context-aware applications, we extract a useful and general research problem: monitoring distance-based anomalies over moving object trajectory streams, and propose data structures and algorithms employing local clustering and piecewise VP-tree based rescheduling to efficiently conduct such a task. The experimental results validate our solutions by showing both the usefulness of distance-based anomalies and orders of magnitude improvement in performance compared to simple pruning approach. To the best of our knowledge, this is the first work that provides efficient support for continuous monitoring of distance-based anomalies over trajectory streams.

ACKNOWLEDGEMENTS: The research of Y. Bu, A. Fu was supported in part by the RGC Earmarked Research Grant of HKSAR CUHK 4120/05E, 4118/06E, and RGC Direct Allocation 07-08. The research of L. Chen is supported by the National Basic Research Program of China (973 Program) under grant No. 2006CB303000, NSFC Project No. 60763001, and NSFC/RGC Joint Research Scheme N_HKUST602 /08. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies. Thanks for Sean X. Wang's insightful comments.

9. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FDOA*, pages 69–104, 1993.

[2] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[3] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. LOF: Identifying density-based local outliers. In *SIGMOD Conference*, pages 93–104, 2000.

[5] A. Bulut and A.K. Singh. SWAT: Hierarchical stream summarization in large networks. In *ICDE*, 2003.

[6] Yuhan Cai and Raymond T. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD Conference*, pages 599–610, 2004.

[7] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *ICDE*, pages 126–133, 1999.

[8] Lei Chen and Raymond T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[9] Lei Chen, M. Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD Conference*, pages 491–502, 2005.

[10] Tzi cker Chiueh. Content-based image indexing. In *VLDB*, pages 582–593, 1994.

[11] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD Conference*, pages 419–429, 1994.

[12] Like Gao, Zhengrong Yao, and Xiaoyang Sean Wang. Evaluating continuous nearest neighbor queries for streaming time series via pre-fetching. In *CIKM*, pages 485–492, 2002.

[13] Pierre Geurts. Pattern extraction for time series classification. In *PKDD*, pages 115–127, 2001.

[14] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artif. Intell. Rev.*, 22(2):85–126, 2004.

[15] Alexander T. Ihler, Jon Hutchins, and Padhraic Smyth. Adaptive event detection with time-varying poisson processes. In *KDD*, pages 207–216, 2006.

[16] Wen Jin, Anthony K. H. Tung, and Jiawei Han. Mining top-n local outliers in large databases. In *KDD*, pages 293–298, 2001.

[17] Eamonn J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[18] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*, pages 151–162, 2001.

[19] Eamonn J. Keogh, Jessica Lin, and Ada Wai-Chee Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *ICDM*, pages 226–233, 2005.

[20] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[21] Edwin M. Knorr, Raymond T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB J.*, 8(3-4):237–253, 2000.

[22] Flip Korn, H. V. Jagadish, and Christos Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD Conference*, pages 289–300, 1997.

[23] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD Conference*, pages 593–604, 2007.

[24] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry A. Kautz. Learning and inferring transportation routines. *Artif. Intell.*, 171(5-6):311–331, 2007.

[25] Junshui Ma and Simon Perkins. Online novelty detection on temporal sequences. In *KDD*, pages 613–618, 2003.

[26] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD Conference*, pages 427–438, 2000.

[27] Yufei Tao, Xiaokui Xiao, and Shuigeng Zhou. Mining distance-based outliers from large databases in any metric space. In *KDD*, pages 394–403, 2006.

[28] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn J. Keogh. Indexing multi-dimensional time-series with support for multiple distance measures. In *KDD*, pages 216–225, 2003.

[29] Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *SIGMOD Conference*, pages 131–142, 2004.

[30] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[31] Qiankun Zhao, Steven C. H. Hoi, Tie-Yan Liu, Sourav S. Bhowmick, Michael R. Lyu, and Wei-Ying Ma. Time-dependent semantic similarity measure of queries using historical click-through data. In *WWW*, pages 543–552, 2006.

[32] Qiankun Zhao, Tie-Yan Liu, Sourav S. Bhowmick, and Wei-Ying Ma. Event detection from evolution of click-through data. In *KDD*, pages 484–493, 2006.

[33] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*, pages 358–369, 2002.

[34] Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD Conference*, pages 181–192, 2003.