Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)

Incremental Local Outlier Detection for Data Streams

Dragoljub Pokrajac
CIS Dept. and AMRC
Delaware State University
Dover DE 19901

Aleksandar Lazarevic
United Tech. Research Center
411 Silver Lane, MS 129-15
East Hartford, CT 06108, USA

Longin Jan Latecki
CIS Department.
Temple University
Philadelphia, PA 19122

***Abstract*. Outlier detection has recently become an important problem in many industrial and financial applications. This problem is further complicated by the fact that in many cases, outliers have to be detected from data streams that arrive at an enormous pace. In this paper, an incremental LOF (Local Outlier Factor) algorithm, appropriate for detecting outliers in data streams, is proposed. The proposed incremental LOF algorithm provides equivalent detection performance as the iterated static LOF algorithm (applied after insertion of each data record), while requiring significantly less computational time. In addition, the incremental LOF algorithm also dynamically updates the profiles of data points. This is a very important property, since data profiles may change over time. The paper provides theoretical evidence that insertion of a new data point as well as deletion of an old data point influence only limited number of their closest neighbors and thus the number of updates per such insertion/deletion does not depend on the total number of points *N* in the data set. Our experiments performed on several simulated and real life data sets have demonstrated that the proposed incremental LOF algorithm is computationally efficient, while at the same time very successful in detecting outliers and changes of distributional behavior in various data stream applications.**

I. INTRODUCTION

Despite the enormous amount of data being collected in many scientific and commercial applications, particular events of interests are still quite rare. These rare events, very often called outliers or anomalies, are defined as events that occur very infrequently (their frequency ranges from 5% to less than 0.01% depending on the application). Detection of outliers (rare events) has recently gained a lot of attention in many domains, ranging from video surveillance and intrusion detection to fraudulent transactions and direct marketing. For example, in video surveillance applications, video trajectories that represent suspicious and/or unlawful activities (e.g. identification of traffic violators on the road, detection of suspicious activities in the vicinity of objects) represent only a small portion of all video trajectories. Similarly, in the network intrusion detection domain, the number of cyber attacks on the network is typically a very small fraction of the total network traffic. Although outliers (rare events) are by definition infrequent, in each of these examples, their importance is quite high compared to other events, making their detection extremely important.

Data mining techniques developed for this problem are based on both supervised and unsupervised learning. Supervised learning methods typically build a prediction model for rare events based on labeled data (the training set), and use it to classify each event [1, 2]. The major drawbacks of supervised data mining techniques include: (1) necessity to have labeled data, which can be extremely time consuming for real life applications, and (2) inability to detect new types of rare events. In contrast, unsupervised learning methods typically do not require labeled data and detect outliers as data points that are very different from the normal (majority) data based on some measure [3]. These methods are typically called outlier/anomaly detection techniques, and their success depends on the choice of similarity measures, feature selection and weighting, etc. They have the advantage of detecting new types of rare events as deviations from normal behavior, but on the other hand they suffer from a possible high rate of false positives, primarily since previously unseen (yet normal) data can be also recognized as outliers/anomalies.

Very often, data in many rare events applications (e.g. network traffic monitoring, video surveillance, web usage logs) arrives continuously at an enormous pace thus posing a significant challenge to analyze it [36]. In such cases, it is important to make decisions quickly and accurately. If there is a sudden or unexpected change in the existing behavior, it is essential to detect this change as soon as possible. Assume, for example, there is a computer in the local area network that uses only limited number of services (e.g., Web traffic, telnet, ftp) through corresponding ports. All these services correspond to certain types of behavior in network traffic data. If the computer suddenly starts to utilize a new service (e.g., ssh), this will certainly look like a new type of behavior in network traffic data. Hence, it will be desirable to detect such behavior as soon as it appears especially since it may very often correspond to illegal or intrusive events. Even in the case when this specific change in behavior is not necessary intrusive or suspicious, it is very important for a security analyst to understand the network traffic and to update the notion of the normal behavior. Further, on-line detection of unusual behavior and events also plays a significant role in video and image analysis [14-16]. Automated identification of suspicious behavior and objects (e.g., people crossing the perimeter around protected areas, leaving unattended luggage at the airport installations, cars driving unusually slow or unusually fast or with unusual trajectories) based on information extracted from video streams is currently an active research area. Other potential applications include traffic control and surveillance of commercial and residential buildings. These tasks are characterized by the need for real-time processing (such that any suspicious activity can be identified prior to making harm to people, facilities and installations) and by dynamic, non-stationary and often noisy environment. Hence, there is necessity for incremental outlier detection that can adapt to novel behavior and provide timely identification of unusual events.

Recently, LOF (Local Outlier Factor) algorithm [9] has been successfully applied in many domains for outlier detection in a batch mode [4, 5]. In this paper, we propose a novel incremental LOF algorithm that is appropriate for detecting outliers in data streams. The proposed incremental LOF algorithm is the first incremental outlier detection algorithm to the best of our knowledge. It provides equivalent detection performance as the static LOF algorithm, and has $O(N\log N)$ time complexity, where N is the total number of data points. The paper shows that insertion of new data points as well as deletion of obsolete points influence only limited number of their nearest neighbors and thus insertion/deletion time complexity per data point does not depend on the total number of points N. Our experiments performed on several simulated and real life data sets have demonstrated that the proposed incremental LOF algorithm can be very successful in detecting outliers in various data streaming applications.

II. BACKGROUND

Outlier detection techniques [40] can be categorized into four groups: (1) statistical approaches; (2) distance based methods; (3) profiling methods; and (4) model-based approaches. In statistical techniques [3, 6, 7], the data points are typically modeled using a stochastic distribution, and points are labeled as outliers depending on their relationship with this model. Distance based approaches [8, 9, 10] detect outliers by computing distances among points. Several recently proposed distance based outlier detection algorithms are based on (1) computing the full dimensional distances among points using all the available features [10] or only feature projections [8]; and (2) on computing the densities of local neighborhoods [9, 35]. In addition, clustering-based techniques have also been used to detect outliers either as side products of the clustering algorithms (points that do not belong to clusters) [11] or as clusters that are significantly smaller than others [12]. In profiling methods, profiles of normal behavior are built using different data mining techniques or heuristic-based approaches, and deviations from them are considered as outliers (e.g., network intrusions). Finally, model-based approaches usually first characterize the normal behavior using some predictive models (e.g. replicator neural networks [13] or unsupervised support vector machines [4, 12]), and then detect outliers as deviations from the learned model.

Initially proposed outlier detection algorithms determine outliers once all the data records (samples) are present in the dataset. We refer to these algorithms as *static outlier detection algorithms*. In contrast, *incremental* outlier detection techniques [38, 39, 41] identify outliers as soon as new data record appears in the dataset. Incremental outlier detection was also used within more general framework of activity monitoring [38]. In addition, Domingos and Hulten [39] proposed broad requirements that incremental algorithms need to meet, while Yamanishi and Takeuchi [41] used on-line discounting distributional learning of Gaussian mixture model and scoring based on the estimated probability density function.

In this study, we use propose an incremental outlier detection algorithm based on computing the densities of local neighborhoods. In our previous work [4], we have experimented with numerous outlier detection algorithms for network intrusion identification, and we have concluded that the local density based outlier detection approach (e.g. LOF) typically achieved the best prediction performance.

The main idea of the LOF algorithm [9] is to assign to each data record a degree of being outlier. This degree is called the *local outlier factor (LOF)* of a data record. Data records (points) with high *LOF* have local densities smaller than their neighborhood and typically represent stronger outliers, unlike data points belonging to uniform clusters that usually tend to have lower *LOF* values. The algorithm for computing the *LOF*s for all data records has the following steps:

1. For each data record q compute ***k*-distance**(q) as distance to the k-th nearest neighbor of q (for definitions, see Section III).
2. Compute reachability distance for each data record q with respect to data record p as follows:

$$\textbf{reach-dist}_k(q,p) = \max(\mathrm{d}(q,p), k\text{-distance}(p)) \quad (1)$$

where $d(q,p)$ is Euclidean distance from q to p.

3. Compute **local reachability density (*lrd*)** of data record q as inverse of the average reachability distance based on the k nearest neighbors of the data record q (In original LOF publication [9], parameter k was named *MinPts*).

$$lrd(q) = \frac{1}{\sum_{p \in kNN(q)} reach-dist_k(q,p)/k}. \quad (2)$$

4. Compute ***LOF*** of data record q as ratio of average local reachability density of q's k nearest neighbors and local reachability density of the data record q.

$$LOF(q) = \frac{\frac{1}{k}\sum_{p \in kNN(q)} lrd(p)}{lrd(q)}. \quad (3)$$

The main advantages of LOF approach over other outlier detection techniques include:

- It detects outliers with respect to density of their neighboring data records; not to the global model.
- It is able to detect outliers regardless the data distribution of normal behavior, since it does not make any assumptions about the distributions of data records.

In order to fully justify the need for **incremental** outlier detection techniques, it is important to understand that applying static LOF outlier detection algorithms to data streams would be extremely computationally inefficient and/or very often may lead to incorrect prediction results. Namely, static LOF algorithm may be applied to data streams in three ways:

1. "Periodic" LOF. Apply LOF algorithm on the entire data set periodically (e.g., after every data block of 1000 data records is inserted, similar to the strategy discussed in [39]) or after all the data records are inserted. The major problem of this approach is inability to detect outliers related to the beginning of new behavior that initially appear within the inserted block. Fig. 1 illustrates this scenario. Assume that a new data point d_n (red asterisk in Fig. 1a) is added to the original data distribution (blue dots in Fig. 1a). Initially, point d_n is an outlier since it is distinct from all other data records. However, when additional data records (red asterisks in Fig. 1b) start to group around the initial data record d_n, these new points are no longer outliers since they

form their own cluster. Ideally, at the time of its insertion, data record d_n should be identified as an outlier [38]. However, in this "periodic" scenario, the LOF algorithm is applied when all data records are added to the original data set and thus already formed the new distribution. Hence, all the points from the new cluster, including d_n, will be identified as normal behavior! On the other hand, if an incremental LOF algorithm is applied after every new data instance is inserted into the data set, not only it is possible to detect points like d_n as outliers but also to detect the moment in time when this change of behavior occurred.

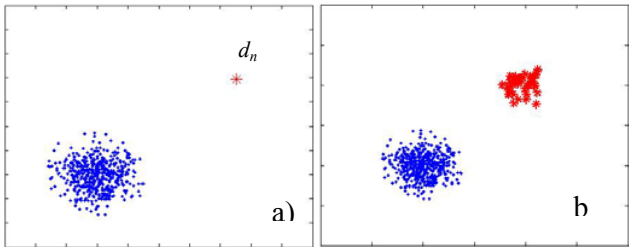


Fig. 1. Inability of static LOF algorithms to identify change of behavior: a) Point d_n can be correctly identified as outlier by "supervised" LOF algorithm but not by "periodic" LOF; b) Points belonging to the new distribution are incorrectly identified as outliers by "supervised" LOF.

2. "Supervised" LOF. Given a training data set D_0 at time interval t_0, LOF algorithm is first applied to compute the *k-distance*s, *lrd* and *LOF* values for all data records from the training data set D_0. For every time interval, $t > t_0$, when a new data record d_n is inserted into the data set, *k-distance*, *reachability distances* and *lrd* values are computed for the new record d_n. However, when computing $LOF(d_n)$ using Eq. (3), $lrd(d_n)$ is used along with pre-computed *lrd* values for the original data set D_0. It is apparent that this approach will result in several problems: (i) Estimated value $LOF(d_n)$ will not be accurate, since it uses pre-computed *k*-distance, *reach-dist* and *lrd* values; (ii) a new behavior (shown as red asterisks in Fig. 1b) will always be detected as outlier since this approach does not update the normal behavior profile; (iii) masquerading (attempt of hiding within existing distribution) cannot be identified, since all inserted data points will always be considered as normal as they belong to normal distribution (Fig. 2). Namely, assume that a new data point d_n (red square in Fig. 2a) is inserted within existing data distribution and all new data points start to group around the point d_n (red squares in Fig. 2b) but with much higher density than the original data distribution. Apparently, these newly added points will form a cluster of very high density which is substantially different than the cluster of the original distribution. "Supervised" LOF approach considers these points to belong to the original data distribution, since it is not aware of new data points forming the dense cluster. On the other hand, incremental LOF algorithm, after insertion of each new data point, would identify this phenomenon, since it can take into account the newly added points, when updating *lrd* and *LOF* values of *existing* points (that are already in the database).

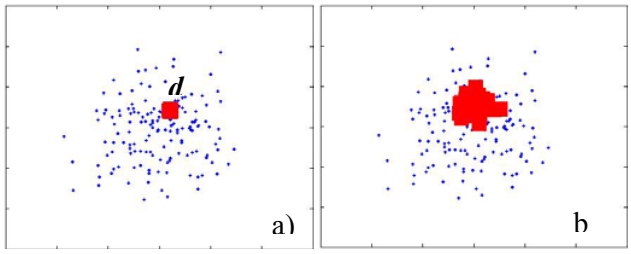


Fig. 2. Detecting masqueraders (hiding within existing distribution)

3. "Iterated" LOF. Re-apply the static LOF algorithm every time a new data record d_n is inserted into the data set. This static LOF algorithm does not suffer from aforementioned problems, but is extremely computationally expensive, since every time a new point is inserted, the algorithm recomputes *LOF* values for *all* the data points from the data set. Knowing that time complexity of LOF algorithm is $O(n \cdot \log n)$ [9], where n is the current number of data records in the data set, total time complexity for this "iterated" approach, after insertion of N points, is :

$$O\left(\sum_{n=1}^{N} n \log n\right) = O(N^2 \cdot \log N), \quad (4)$$

Our proposed incremental LOF algorithm is designed to provably provide the same prediction performance (detection rate and false alarm rate) as the "iterated" *LOF*. It is achieved by consistently maintaining for all existing points in the database the same *LOF* values as the "iterated" LOF algorithm. Our proposed incremental LOF algorithm efficiently addresses the problems mentioned in Fig. 1 and 2, but has time complexity $O(N \cdot \log N)$ thus clearly outperforming the static "iterated" LOF approach. After all N data records are inserted into the data set, the final result of the incremental LOF algorithm on N data points is independent of the order of insertion and equivalent to the "periodic" LOF executed after all the data records are inserted.

III. METHODOLOGY

When designing incremental LOF algorithm, we have been motivated by two goals. First, the result of the incremental algorithm must be equivalent to the result of the "static" algorithm every time t a new point is inserted into a data set. Thus, there would not be a difference between applying incremental LOF and the "periodic" static LOF when all data records up to time instant t are available. Second, asymptotic time complexity of incremental LOF algorithm has to be comparable to the static LOF algorithm. In order to have feasible incremental algorithm, it is essential that, at any time moment t, insertion/deletion of the data record results in limited (preferably small) number of updates of algorithm parameters. Specifically, the number of updates per each insertion/deletion must not be dependent on the current number of records in the dataset; otherwise, the performance of the incremental LOF algorithm would be $\Omega(N^2)$ where N is the final size of the dataset. In this section, we demonstrate efficient insertion and deletion of records in the incremental *LOF* algorithm and provide its exact time complexity analysis.

A. Incremental LOF algorithm

The proposed incremental LOF algorithm computes *LOF* value for each data record inserted into the data set and instantly determines whether inserted data record is outlier. In addition, *LOF* values for existing data records are updated if needed.

i. Insertion. In the insertion part, the algorithm performs two steps: a) insertion of new record, when it computes *reach-dist*, *lrd* and *LOF* values of a new point; b) maintenance, when it updates *k*-distances, *reach-dist*, *lrd* and *LOF* values for affected existing points. Let us first illustrate these steps through the example of inserting a new data point *n* into a data set shown on Fig. 3a. If we assume $k = 2$, we first need to compute reachability distances to two nearest neighbors of the new data point *n* (data points 4, 6 in Fig. 3a), so that its *lrd* value can be computed. As it is shown further in the paper (Theorem 1), insertion of the point *n* may decrease the *k-distance* of certain neighboring points, and it can happen only to those points that have the new point *n* in their *k*-neighborhood. Hence, we need to determine all such affected points, (points 1, 3, 4, 6 have point *n* in their 2-neighborhood, see Fig. 3a). According to Eq. (1), when *k-distance*(*p*) changes for a point *p*, *reach-dist*(*q*,*p*) will be affected only for points *q* that are in *k*-neighborhood of the point *p*. In our example, previous 2-neighbors of data point 3 are the data points 2, and 11, so *reach-dist*(11,3), and *reach-dist*(2,3) will be updated (Fig. 3b). According to Eq. (2), *lrd* value of a point *q* is affected if: a) the *k*-neighborhood of the point *q* changes or b) *reach-dist* from point q to one of its k-neighbors changes. The 2-neighborhood of a point will change only if the new point *n* becomes one of its 2-neighbors. Hence, we need to update *lrd* on all points to which the point *n* is now one of their 2-neighbors (points 1, 3, 4, 6 in Fig. 3b) and on all points *q* where *reach-dist*(*q*,*p*) is updated and *p* is among 2-nearest neighbors of *q* (points 2,5,7 in Fig. 3c). According to Eq. (3), *LOF* values of an existing point *q* should be updated if *lrd*(*q*) is updated (points 1,2,3,4,5,6,7 in Fig. 3d) or *lrd*(*p*) of one of its 2-neighbors *p* changes (points 8,9,10 In Fig 3d). Note that *LOF* value of point 11 is not updated since point 3 (where *lrd* is updated) is not among its 2 nearest neighbors.

The general framework for the incremental LOF method is shown in Fig. 4. As in the static LOF algorithm [9], we define *k-th* nearest neighbor of a record *p* as a record *q* from the dataset *S* such that for at least *k* records $o' \in S \setminus \{p\}$ it holds that $d(p,o') \le d(p,q)$, and for at most *k*-1 records $o' \in S \setminus \{p\}$ holds that $d(p,o') < d(p,q)$, where *d(p,q)* denotes Euclidean distance between data records *p* and *q*. We refer $d(p,q)$ as *k-distance*(*p*). *k* nearest neighbors (referred to as *k*NN(*p*)) include all points $r \in S \setminus \{p\}$ such that $d(p,r) \le d(p,q)$. We also define *k reverse* nearest neighbors of *p* (referred to as *kRNN* (*p*)) as all points *q* for which *p* is among their *k* nearest neighbors. For a given data record *p*, *k*NN(*p*) and *k*RNN(*p*) can be respectively retrieved by executing nearest-neighbor and reverse (a.k.a. inverse) nearest neighbor queries [17-20,42]on a dataset *S*. The correctness of the insertion algorithm is based on the following Theorems 1-4.

Theorem 1. The insertion of point p_c affects the *k*-distance at points p_j that have point p_c in their *k-nearest* neighborhood, i.e., where $p_j \in k\text{RNN}(p_c)$. New *k*-distances of the affected points p_j are updated as follows:

$$k-\text{distance}^{(new)}(p_j)=\begin{cases} d(p_j,p_c), & p_c \text{ is the } k\text{ - th nearest neighbor of } p_j \\ (k-1)\text{distance}^{(old)}(p_j), & \text{otherwise.} \end{cases} \quad (5)$$

Proof. (sketch). In insertion, *k-distance* of an existing point p_j changes when a new point enters the *k*-th nearest neighborhood of p_j, since in this case the *k*-neighborhood of p_j changes. If a new point p_c is the new *k*-th nearest neighbor of p_j, its distance from p_j becomes the new *k-distance*(p_j). Otherwise, old *k*-1*th* neighbor of p_j becomes the new *k*-th nearest neighbor of p_j (see Fig. 5).

Corollary 1. During insertion, k-distance cannot increase, i.e., $k-\text{distance}^{(new)}(p_j) \le k-\text{distance}^{(old)}(p_j)$. □

Theorem 2. Change of *k*-distance(p_j) may affect *reach-dist*$_k$ (p_i,p_j) for points p_i that are *k*-neighbors of p_j. (see Fig 5b).

Proof (sketch). Using (1), $\forall\ p_i\ d(p_i, p_j) > k\text{-distance}^{(\text{old})}(p_j)$, $\Rightarrow$ *reach-dist*$_k^{(\text{old})}$ (p_i,p_j)= $d(p_i, p_j)$. According to *Corollary* 1, *k*-distance(p_j) cannot increase, hence if $d(p_i, p_j) > k\text{-distance}^{(\text{old})}(p_j)$, *reach-dist*$_k^{(\text{new})}$ (p_i,p_j) = *reach-dist*$_k^{(\text{old})}$ (p_i,p_j). □

Theorem 3. lrd value needs to be updated for every record (denoted with p_m in Fig. 4) for which its *k*-neighborhood changes or for which reachability distance to one of its *k*NN changes. Hence, after each update of *reach-dist*$_k$ (p_i,p_j) we have to update *lrd*(p_i) if p_j is among *k*NN(p_i). Also, *lrd* is updated for all points p_j whose *k*-distance was updated.

Proof (sketch). Change of *k*-neighborhood of p_m affects the scope of the sum in Eq. (2) computed for all *k*-neighbors of p_m. Change of the reachability distance between p_m and some of its *k-nearest* neighbors affects corresponding term in the denominator of Eq. (2). □

Theorem 4. LOF value needs to be updated for all data records p_m which *lrd* has been updated (since *lrd*(p_m) is a denominator in Eq. (3)) and for those records that have records p_m in their *k*NNs. Hence, the set of data records where *LOF* needs to be updated (according to (3)) corresponds to union of records p_m and their *k*RNN.

Proof (sketch). Similar to the proof of *Theorem 3*, using (3). □

ii. Deletion. In data stream applications it is sometimes necessary to delete certain data records (e.g., due to their obsoleteness). Very often, not only a single data record is deleted from the data set, but the entire data block that may correspond to particular outdated behavior. Similarly like in an insertion, upon deleting the block of data records S_{delete}, there is a need to update parameters of the *incremental* LOF algorithm.

The general framework for deleting the block of data records S_{delete} from the dataset *S* is given in Fig. 6. The deletion of each record $p_c \in S_{delete}$ from dataset *S* influences the *k*-distances of its *k*RNN. *k*-neighborhood increases for each data record p_j that is in reverse *k*-nearest neighborhood of p_c. For such records, *k*-distance(p_j) becomes equal to the distance from p_j to its new *k*-th nearest neighbor. The reachability distances from p_j's (*k*-1) nearest neighbors p_i to p_j need to be updated. Observe that the reachability distance from the *k*-th neighbor of p_j to record p_j is already equal to their Euclidean distance

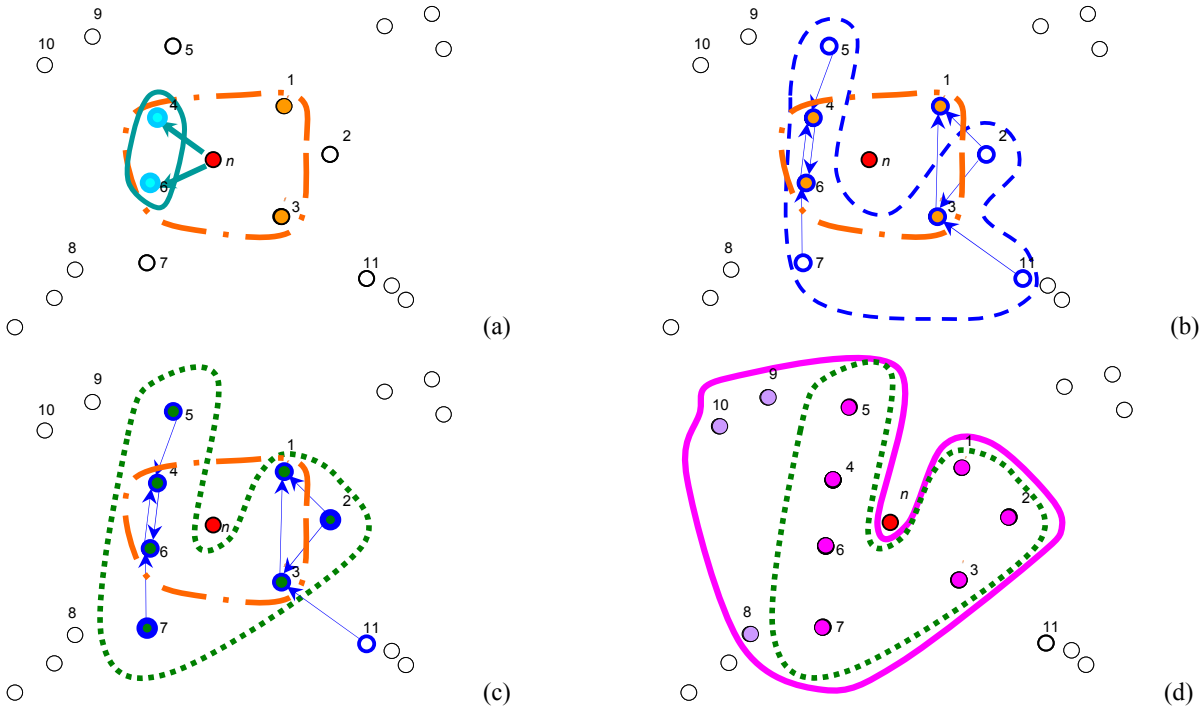


Fig. 3. The illustration of the proposed incremental LOF algorithm. a) Insertion of a new point *n* (red) results in computation of the reachability distance to its two nearest neighbors 4, 6 (cyan) and to update of 2-distance to reverse nearest neighbors of n (1,3,4,6, yellow). b) Reachability distance reach-dist(q,p) is updated to all reverse k-neighbors of the point *n* from their 2-neighbors (blue arrows from q to p). c) lrd is updated for all points where 2-distance is updated and for points which reachability distance to their 2-neighbor changes (green). d) *LOF* is updated for points where lrd is updated and for points where lrd of one of their 2-neighbors is updated (pink).

Incremental LOF_insertion(*Dataset S*)
- Given: Set S $\{p_1, \ldots, p_N\}$ $p_i \in \mathbf{R}^D$, where D corresponds to the dimensionality of data records.
- For each data point p_c in data set S
  - insert(p_c)
  - *Compute* kNN(p_c)
  - ($\forall p_j \in$ kNN(p_c))
    compute *reach-dist*$_k$(p_c,p_j) using Eq. (1);

  //Update_neighbors of p_c
  - $S_{update\_k\_distance}$ =kRNN(p_c);
  - ($\forall p_j \in S_{update\_k\_distance}$)
    update k-distance(p_j) using Eq.(5);
  - $S_{update\_lrd} = S_{update\_k\_distance}$;
  - ($\forall p_j \in S_{update\_k\_distance}$), ($\forall p_i \in k\text{NN}(p_j) \setminus \{p_c\}$)
    reach-dist$_k$(p_i,p_j) =k-distance(p_j);
    if $p_j \in k\text{NN}(p_i)$
    $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
  - $S_{update\_LOF} = S_{update\_lrd}$;
  - ($\forall p_m \in S_{update\_lrd}$)
    update lrd(p_m) using Eq. (2);
    $S_{update\_LOF} = S_{update\_LOF} \cup k\text{RNN}(p_m)$;
  - ($\forall p_i \in S_{update\_LOF}$)
    update LOF(p_i) using Eq.(3);
  - compute lrd(p_c) using Eq.(2);
  - compute LOF(p_c) using Eq.(3);
- End //for

Fig. 4. The general framework for insertion of data record and computing its *LOF* value in incremental LOF algorithm.

d(p_i, p_j) and does not need to be updated (Fig. 7). Analog to insertion, *lrd* value needs to be updated for all points p_j where *k*-distance is updated. In addition, *lrd* value needs to be updated for points p_i such that p_i is in *k*NN of p_j and p_j is in *k*NN of p_i. Finally, *LOF* value is updated for all points p_m on which *lrd* value is updated as well as on their *k*RNN. The correctness of the deletion algorithm can be proven analog to the correctness of the insertion algorithm.

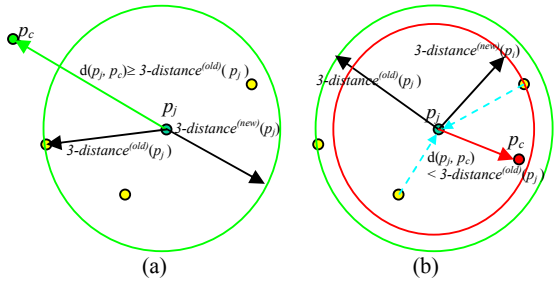


Fig. 5. Update of k-nearest neighbor distance upon insertion of a new record (*k*=3). a) New record p_c is not among 3-nearest neighbors of record $p_j \Rightarrow$ *3-distance*(p_j) does not change; b) New record p_c *is* among 3-nearest neighbors of $p_j \Rightarrow$ *3-distance*(p_j) decreases. Cyan dashed lines denote updates of reachability distances between point p_j and two old points.

B. Computational efficiency of the incremental LOF algorithm

To determine time complexity of the proposed incremental LOF algorithm, it is essential to demonstrate that the number of affected data records (updates of *k*-distance, reachability distances, *lrd* and *LOF values*) does not depend on the current number *n* of records in the dataset, as stated by Theorems 5-8. Subsequently, Corollaries 3-5 provide asymptotic time complexity for the proposed algorithm.

Incremental LOF deletion(*Dataset* S,S_{delete})
- $S_{update\_k\_distance}=\emptyset$;
- $(\forall p_c \in S_{delete})$
  $S_{update\_k\_distance} = S_{update\_k\_distance} \cup kRNN(p_c)$;
  delete(p_c); //we can delete p_c after finding
  // all reverse neighbors
- $S_{update\_k\_distance} = S_{update\_k\_distance} \setminus S_{delete}$; //points from S_{delete} may still be present when computing reverse k-nearest neighbors
- $(\forall p_j \in S_{update\_k\_distance})$
  update k-distance(p_j);
- $S_{update\_lrd} = S_{update\_k\_distance}$;
- $(\forall p_j \in S_{update\_k\_distance})$ $(\forall p_i \in (k\text{-}1)NN(p_j))$
  $reach\text{-}dist_k(p_i,p_j)$=k-distance($p_j$);
  if $p_j \in kNN(p_i)$
  $S_{update\_lrd} = S_{update\_lrd} \cup \{p_i\}$;
- $S_{update\_LOF} = S_{update\_lrd}$;
- $(\forall p_m \in S_{update\_lrd})$
  update $lrd(p_m)$ using Eq. (2);
  $S_{update\_LOF} = S_{update\_LOF} \cup kRNN(p_m)$;
- $(\forall p_i \in S_{update\_LOF})$
  update $LOF(p_i)$ using Eq.(3);

return

Fig. 6. The framework for deletion of data record in *incremental* LOF method.

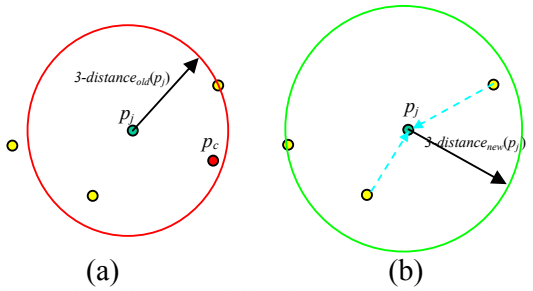


Fig. 7. Update of k-nearest neighbor distance upon deletion of record p_c (k=3). a) Prior to deletion, data record p_c is among 3-nearest neighbors of record p_j; b) After deletion of p_c, *3-distance*(p_j) increases and reachability distances from two nearest neighbors of p_j (denoted by cyan dashed lines) are updated to *3-distance*(p_j).

Theorem 5. Maximal number F of k reverse nearest neighbors of a record p is proportional to k, exponentially proportional to the dimensionality D, and does not depend on n.

To prove Theorem 5, we will first establish a few definitions [37] and prove Lemmas 1, 2 necessary to establish D-dimensional geometry of the space where the considered data records reside.

Definition 1. The cone C in D-dimensional space with vertex v and axis l is locus of points x such that the Euclidean distance of the point x to the vertex d(x,v) is proportional to the distance d(x',v) of point's projection x' onto l to the vertex v. A line containing a point x on cone and the origin is called *generatix*. When the vertex is at the origin of the coordinate system (v=O) we refer a cone as *centered cone*.

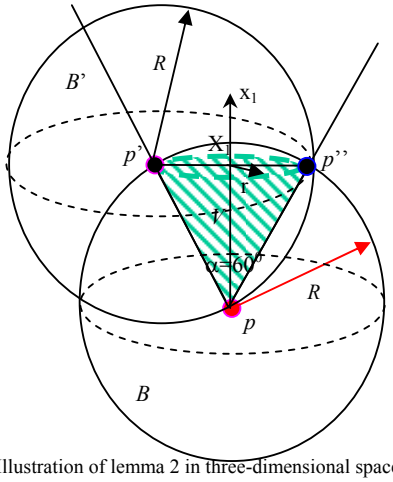


Fig. 8. Illustration of lemma 2 in three-dimensional space

Definition 2. A half-axis angle of a cone is an angle $\alpha/2=\angle xvx'$. It is angle between the axis and any generatrix.

Lemma 1. Coordinates X_1, X_2,…, X_D of a point x on the centered cone C, where the axis l of the cone is parallel to the x_1 axis of the coordinate system, satisfy:

$$X_2^2+X_3^2+\ldots X_D^2 = a^2 X_1^2,\ a>0 \qquad (6)$$

where a is a pre-specified parameter. Half-axis angle of C satisfies the following condition:

$$\cos\frac{\alpha}{2}=\frac{X_1}{\sqrt{\sum_{i=1}^{D} X_i^2}}=\frac{1}{\sqrt{1+a^2}}. \qquad (7)$$

Proof. Follows directly from definitions 1 and 2 and the fact that the length of x' is X_1 when l is parallel to x_1. □

Definition 3. All points which coordinates satisfy relation $X_2^2+X_3^2+\ldots X_D^2 < a^2 X_1^2$ are *inside* the cone C and comprise set *inside*(C).

Definition 4. [36] The ball $B(c,R)$ in D-dimensional space is locus of points x such that the distance of the point x from a prespecified point c is smaller or equal R>0.

Lemma 2. Consider cone C with vertex p and half-axis angle $\alpha/2 \le 30^0$. Consider ball $B(p, R)$, and volume $V=B\cap C$. $(\forall p' \in C)$ $d(p',p)=R \Rightarrow V \subset B'(p',R)$ (See Fig. 8 for tree-dimensional illustration).

Proof. (sketch). Without loss of generality, we may assume that the cone is centered, i.e., that p is at the origin. Let coordinates of point p' be X'_1, X'_2,…, X'_D. Consider a point p'' symmetric to p' with respect to the x_1 axis. The point p'' has coordinates X'_1, $-X'_2$,…, $-X'_D$. It is easy to observe that $p'' \in C$ and $d(p,p'')=R$. The distance between these two points is $d(p',p'')=2\sqrt{X_2^2+\ldots X_D^2}=2aX_1$. Since points p', p'' are on the sphere with radius R, from Eq. (7) we can obtain $d(p',p'')=2R\sin\frac{\alpha}{2} \le 2R\sin 30^0 = R$. Therefore, the ball $B'(p'$, R) contains the point p'' antipodal to p'. It can be shown that B' also contains any other boundary point of V. □

Definition 5. A frame of cones C_i, i = 1,…,h is defined as a set of h cones with the common vertex p and common angle

α. The frame of cones completely cover the *D*-dimensional Euclidean space E^D if: $\bigcup_{i=1}^{h} C_i = E^D$ (see Fig. 9).

Lemma 3. The lower bound[1] for the number of $\alpha = 60^0$ cones in a *D*-dimensional frame is $h_{\min} = \frac{\int_0^{\pi} \sin^{D-2}\varphi_1 d\varphi_1}{\int_0^{\pi/6} \sin^{D-2}\varphi_1 d\varphi_1} = \Theta\left(2^D\sqrt{D}\right)$.

Proof (sketch). The lower bound of the number of cones in a frame is equal to the ratio of the area of the hypersphere and the area of the hyperspherical cap (part of the hypersphere inside the cone) with angle α. Details are presented in [21]. Note that h_{min} depends only on geometry of *D*-dimensional space and is independent of the number or the placement of *D*-dimensional data records. □

The following Definition 6 and Corollary 2 link geometry of *D*-dimensional cones to proximity notion in *D*-dimensional datasets.

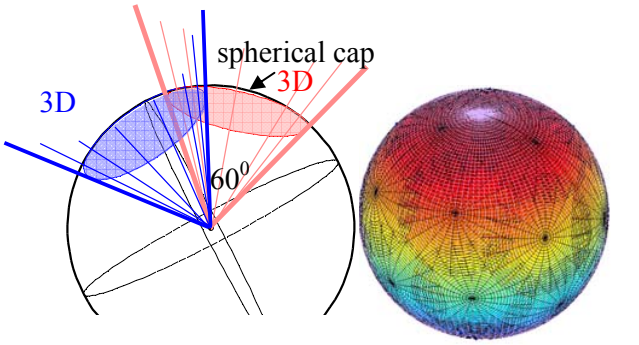
Definition 6. Let *S'* be set of the *D*-dimensional data records inside a cone *C* with vertex *p*. *k*-nearest neighbor of point *p in the cone C* is a record *q* from the dataset *S'* such that for at least *k* records $o' \in S' \setminus \{p\}$ it holds that $d(p,o') \le d(p,q)$, and for at most *k*-1 records $o' \in S' \setminus \{p\}$ holds that $d(p,o') < d(p,q)$. We also define *k-distance*$_C(p)$ as a distance from record *p* to its *k*-th nearest neighbor *q* in the cone. *k*-nearest neighborhood of *p* in cone *C*, (referred to as $kNN_c(p)$) includes all records $r \in S' \setminus \{p\}$ such that $d(p,r) \le d(p,q)$.

Corollary 2. Consider cone *C* centered at data point *p* with $\alpha \le 60^0$. Let *p'* be a data point in cone *C* .

$$(\forall p' \in inside(C)) p' \notin kNN_C(p) \Rightarrow p \notin kNN_C(p')$$

Proof. (sketch). Consider ball *B*(*p*, *k*-distance$_C(p)$). According to the Definition 6, the volume $V = B \cap C$ contains exactly *k* points *other* than *p*. Consider data point *p'* such that *d*(*p*,*p'*) > *k*-distance$_C(p)$. Consider now ball *B'*(*p'*, *k*-distance$_C(p)$). According to Lemma 2, this ball has as a subset the whole volume *V* that contains total of *k* + 1 data points (*including p*). Hence, *k*-distance$_C(p')$< *k*-distance$_C(p)$. □

Proof of Theorem 5. Due to Corollary 2 and Definitions 5, 6, $kRNN(p) \subset \bigcup_{i=1}^{h} kNN_{C_i}(p)$, where C_i, i=1,…,*h* is frame of cones with vertex *p*. Hence, due to Lemma 3 and Definition 6, $|kRNN(p)| \le F = |kNN_{C_i}(p)| \cdot h_{\min} = \Theta\left(k 2^D \sqrt{D}\right)$. Since neither $h_{\min}$ nor *k* depend on *n*, the number of reverse nearest neighbors does not depend on the current number of points in the dataset. □



(a) (b)

Fig. 9. (a) Two 60-degree 3D cones (b) Spherical caps of the frame consisting of 22 cones that completely cover 3D space.

The following theorems provide the upper bound for the number of points where *k*-distance, *lrd* and *LOF* are updated.

Theorem 6. The number of data records where *k*-distance needs to be updated is $|S_{update\_k\_distance}| \le F$ for insertion, and $|S_{update\_k\_distance}| \le F^*|S_{delete}|$, for deletion block of size $|S_{delete}|$.

Proof (sketch). For insertion/deletion of one data record *s*, *k*-distance needs to be updated on all data records that have the inserted/deleted data record in its neighborhood, i.e., on all reverse nearest neighbors of *s*. Theorem 5 bounds the number of reverse neighbors with *F*. □

Theorem 7. Number of data records where *lrd* is updated is $|S_{update\_lrd}| \le k \cdot |S_{update\_k\_distance}|$.

Proof (sketch). *lrd* values are updated on points from $|S_{update\_k\_distance}|$ and *may* be updated on their *k*-nearest neighbors.□

Theorem 8. Number of data records where *LOF* is updated is $|S_{update\_LOF}| \le (1+F) \cdot |S_{update\_lrd}|$.

Proof (sketch). *LOF* value is updated on data records from $|S_{update\_lrd}|$ and their reverse nearest neighbors, thus giving the bound stated in the Theorem 8. □

The following corollaries provide asymptotic time complexity for the proposed algorithm.

Corollary 3. The asymptotic time complexity for insertion and deletion in incremental *LOF* is[2]:

$$T_{incrLOF\_ins} = O(k \cdot F \cdot T_{kNN} + k \cdot F \cdot T_{kRNN} + F^2 \cdot k + T_{insert}), \quad (8)$$

$$T_{incrLOF\_del} = O(|S_{delete}| \cdot (k \cdot F \cdot T_{kNN} + k \cdot F \cdot T_{kRNN} + F^2 \cdot k + T_{delete})).$$

Here, T_{kNN} and T_{kRNN} are time complexities of *k*NN and *k*RNN algorithms respectively, while T_{insert} and T_{delete} correspond to time needed for the insertion and deletion of a data record into/from the database (including index updating).

Proof (sketch). Follows from the algorithms for insertion and deletion in incremental LOF, given in Fig. 4 and Fig. 6 respectively, and Theorems 6-8. □

[1] Suboptimal values for *h* can be obtained by techniques that construct spherical codes [22], followed by covering verification (to ensure that the space around *p* is completely covered), e.g., based on examination of convex hulls facets [23]. Analog to the problem of optimal spherical codes, the problem of finding the smallest possible *h* for arbitrary *D* is unresolved and is related (but *not* equivalent) to sphere covering problem [22]. Using the aforementioned suboptimal construction, in [21] the upper bound on *h* is shown for several dimensions: more precisely, *h* is demonstrated to have upper bound of 22 (see Fig. 8b), 85, 305 in R^3, R^4, R^5, correspondingly

[2] By maintaining list of *kRNN(p)* for each record *p*, the time complexities can be further reduced to:
$T_{\text{incrLOF\_ins}} = O(k \cdot F \cdot T_{kNN} + T_{kRNN} + F^2 \cdot k + T_{\text{insert}})$;
$T_{\text{incrLOF\_del}} = O(|S_{\text{delete}}| \cdot (k \cdot F \cdot T_{kNN} + F^2 \cdot k + T_{\text{delete}}))$

Corollary 4. When efficient algorithms for kNN [e.g. 24], kRNN [e.g., 17-20,42], as well as efficient indexing structures for inserting/deleting the records [25, 26] are used (where $T_{kNN} = T_{kRNN} = T_{insert} = T_{delete} = O(\log n)$, the time complexities of $T_{incrLOFins}$ and $T_{incrLOFdel}$ are logarithmic in the current size n of the database, e.g.,:

$$T_{incrLOFins} = O(k \cdot F \cdot \log n + F^2 \cdot k). \quad (9)$$

Corollary 5. Time complexity of the incremental LOF algorithm after all updates to the dataset of size N are applied is $O(N \cdot \log N)$.

Proof. Directly follows from Corollary 4. □

Note that according to Theorem 5, the time complexity of the *incremental* LOF may exponentially increase with the dimensionality D. However, this is well-known problem of static LOF [9] as well as other density based algorithms and not a particular issue with incremental LOF.

IV. EXPERIMENTAL RESULTS

Our experiments were performed on several synthetic data and real life data sets. In all our experiments, we have assumed that we have information about the outliers in the data set, so we could evaluate the detection performance. In the following subsectios we evaluate time complexity (subsection A) and outlier detection accuracy (subsections B, C) with respect to ground truth outlier information.

A. Time Complexity Analysis

Our time complexity analysis was performed on synthetic data sets, since we could better control the total number of data records N in the data set as well as the number of dimensions D. Reported experimental results provide evidence about (i) relation between the number of updates for *LOF* values and the total number of data points N; (ii) the dependence of the number of updates for *LOF* values on *LOF* parameter k; and (iii) the dependence of the number of updates for *LOF* values on the dimension D.

Our synthetic data sets had different number of data records ($N \in \{100,200,\ldots,5000\}$), as well as different number of dimensions, ($D \in \{2,3,4,5,10\}$). For each pair (D, N), we have created 100 data sets with N random records generated from D-variate distribution. We experimented with both uniform and standard (zero mean, unit covariance matrix) Gaussian distribution. For each of 100 data sets generated for the pair (D, N), we varied the values of the parameter k (5, 10, 15, 20) of the algorithm and then measured the number of updates for *k-distance*, *reach-dist, lrd* and *LOF values* in the incremental LOF algorithm for insertion of a new data record into the dataset. Here, for each pair (D, N) we report *average* number of *LOF* updates for all 100 data sets generated using the standard Gaussian distributions. Results obtained for the data sets generated using uniform distribution are analog and not reported here due to lack of space.

Fig. 10 shows how the number of updates of *LOF* values depends on the total number of data records N (x-axis in Fig. 10) for different number of dimensions D (different lines in graphs in Fig. 10), where each graph corresponds to distinct value of parameter k.

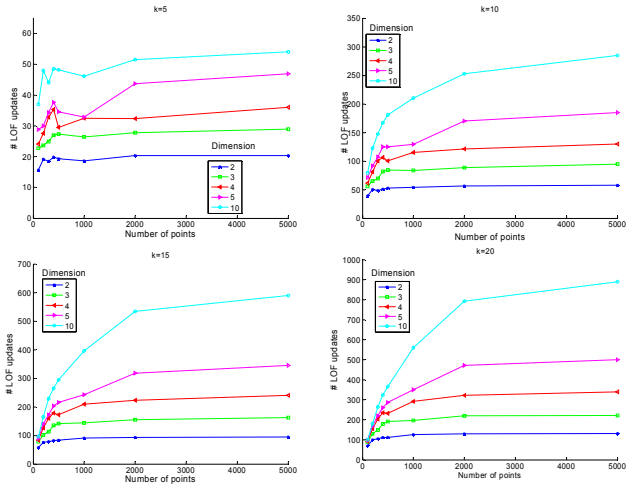


Fig. 10. The dependence of number of *LOF* updates on the database size N for different number of dimensions D and different values of parameter k. The results are averaged over 100 data sets generated from standardized Gaussian distribution.

Analyzing Fig. 10, it can be observed that the number of updates of *LOF* value stabilizes for sufficiently large N, which is in accordance with our theoretical analysis from section III.B. showing $O(1)$ updates with respect to N. It is interesting to note that for larger k, the number of data records, necessary to show stabilization of number of *LOF* updates, is generally larger. However, for typically used values of k (5-20) [4,5] the number of *LOF* updates becomes constant for N>5000.

Fig. 11 shows the average number of *LOF* updates vs. parameter k (each curve corresponds to a particular value of D) on database of N = 2000 points. The left graph contains abscise in linear scale, while the abscise in the right graph is *quadratic* (proportional to k^2). Fig. 11 shows that the *actual* number of updates seems to change not faster than k^2. Therefore, the worst-case upper bound $O(F^2 \cdot k)$ =$O(k^3)$ obtained in Section III.B. seems to be rather pessimistic. This phenomenon is due to the fact that in reality, not all updates of *reach*-dist values result in update of *lrd* value. Also, a data record may belong to reverse nearest neighbors of multiple records on which *lrd* has been updated. Hence, such data record, although output from several *k*RNN queries, will result only in one update of *LOF* value.

Fig. 11 also provides an insight on the dependence of the number of *LOF* updates on the number of dimensions D. While undoubtedly the number of *LOF* updates increases with D, it was difficult to confirm (or reject) theoretical upper boundary of the exponential dependence (see Section III.B). However, it is evident that the growth of *LOF* updates with respect to dimensionality D is *not* explosive (the average number of updates stay bellow 1000 even for D=10, k=20). One of the reasons is that considered upper bound for the number of reverse neighbors is the worst case and is reached rather infrequently. Hence, we anticipate that the dimensionality of the data will not become the bottleneck of the incremental LOF algorithm due to number of *LOF* updates, but rather due to inability of indexing structures to resist curse of dimensionality [27].

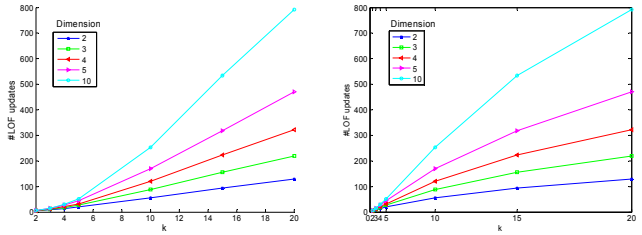


Fig. 11. The dependence of number of *LOF* updates on the value of parameter *k* for different number of dimensions *D*. The results are averaged over 100 datasets with 2000 records generated from standardized normal distribution. a) Linear abscise; b) Abscise proportional to k^2.

B. Learning New Behavior

Our first synthetic data set used to analyze ability of incremental LOF to learn new behavior in a non-stationary data corresponded to 1000 data records generated from 2-modal mixture of 2-dimensional Gaussian distributions with *different means*. The data set consisted of 500 records generated from Gaussian distribution $N(\boldsymbol{\mu}_1,\Sigma_1)$ and 500 records with Gaussian distribution $N(\boldsymbol{\mu}_2,\Sigma_2)$, where $\boldsymbol{\mu}_1=[-1\,-1]$, $\boldsymbol{\mu}_2=[+1\,+1]$, $\Sigma_1=\Sigma_2=\begin{bmatrix}0.1 & 0\\ 0 & 0.1\end{bmatrix}$.

Fig.12 shows current *LOF* value after inserting data records 501, 505, 510 and 1000. At n=501, we identified inserted record from a new distribution as an outlier as soon as it appeared. However, when the "switch" to the new distribution was complete, the incremental LOF learned the new distribution as a part of regular behavior (starting from n=510) so the new records were correctly labeled as normal. As discussed in Section II, if we used "supervised" static LOF (trained at n=500), the data records from the new distribution will *always* be marked as outliers, since they did not appear in original data on which the "supervised" static LOF was trained (see Fig. 1). After all records were inserted, the result (*LOF* value for n=1000) was identical to *LOF* values obtained using the "periodic" static LOF algorithm. However, the "periodic" LOF algorithm will identify all records belonging to the new distribution as normal (although some of them were outliers at the time of insertion).

The second synthetic data set consisted of 1000 data records belonging to 2-modal 2-dimensional Gaussian mixture with same mean but *different variances*. This data set was created to illustrate the attempt of *masquerading* (hiding within existing distribution, see Fig. 2). The data set had 500 data records of Gaussian distribution $N(\boldsymbol{\mu}_1,\Sigma_1)$ and 500 data records with Gaussian distribution $N(\boldsymbol{\mu}_2,\Sigma_2)$, where

$$\boldsymbol{\mu}_1=\boldsymbol{\mu}_2=\boldsymbol{\mu}=[0\quad 0],\Sigma_1=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix},\Sigma_2=\begin{bmatrix}0.0001 & 0\\ 0 & 0.0001\end{bmatrix}.$$

Fig.13 shows current *LOF* values obtained using the incremental LOF algorithm, after insertion of records 501, 515, 600 and 1000. Initially (at n=501), the computed *LOF* values do not reveal any change of behavior. However, while additional data records continue to appear within a new distribution, their *lrd* values will start to increase (larger local density of the new distribution). According to Eq. (3) this fact will cause increase of *LOF* values for points from the *old* distribution that are on the borders of the new distribution (e.g., for n=600, the maximal *LOF* value becomes larger than 10). Since incremental LOF algorithm keeps track of updates of *LOF* values for existing data records (already in the database), this phenomenon is easy to identify. Therefore, it is apparent that the incremental LOF algorithm is capable of identifying the masquerading attempt as well as an approximate time when the attempt begins! As already discussed in Section II, "supervised" static LOF (trained at n=500 on records from the first distribution) is not capable of identifying this behavior, since new data records (corresponding to new, much denser distribution) are not considered when computing *lrd*.

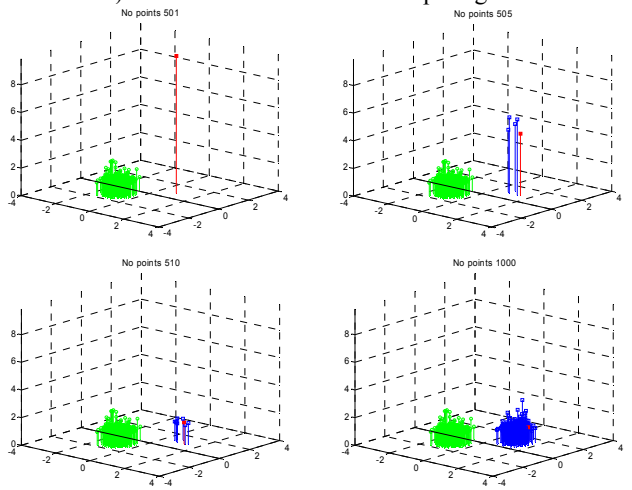


Fig. 12. The result of incremental LOF algorithm (k=10) on dataset consisting of 500 records of Normal distribution $N(\mu_1,\Sigma)$ followed by 500 records with Normal distribution $N(\mu_2,\Sigma)$. $\mu_1=[-1\,-1],\mu_2=[+1\,+1],\Sigma=diag(0.1,0.1)$ after a) n=501; b) n=505; c) n=510; d) n=1000 points inserted. New data records (inserted at time instant *n)* are marked *red*.

In contrast, "periodic" LOF method will be able to identify the masquerading, but this identification will be delayed for the period of *LOF* update.

C. Experiments on real life data sets

To illustrate the ability of incremental LOF algorithm to identify outliers in dynamic environment, we first selected two real life data sets containing video sequences. The first data set is composed of 100 video frames (data is available at: www.cis.temple.edu/~latecki/TestData/SimTest.zip). The features from video frames are extracted using the procedure described in [28]. Our goal was to identify sudden changes in selected video frames, which is important problem in video analysis [16], especially in analysis of streaming videos (e.g., video-surveillance). Analyzing Fig. 14, it can be observed that the proposed incremental LOF correctly detected all the sudden changes in video frames, while not producing any false alarm. These changes were caused by the appearance of a new object (frame 21), zooming objects to camera (frame 31) and novel video content (frames 41, 61, 71, 91). On the other hand, static "periodic" LOF algorithm computed after all 100 frames did not detect any of these frames as outliers, while

"supervised" LOF algorithm had very large false alarm rate due to data non-stationarity.

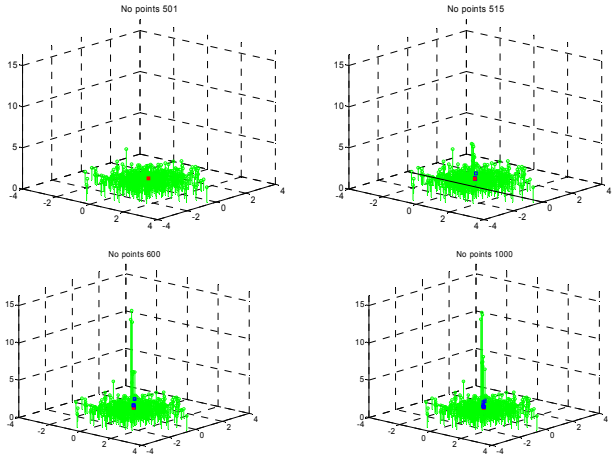


Fig. 13. The result of incremental LOF algorithm (k=10) on dataset consisting of 500 recordsaussian distribution $N(\boldsymbol{\mu},\boldsymbol{\Sigma}_1)$ and 500 records with Gaussian distribution $N(\boldsymbol{\mu},\boldsymbol{\Sigma}_2)$. $\boldsymbol{\mu}$=[0 0], $\boldsymbol{\Sigma}_1$=*diag*(1,1), $\boldsymbol{\Sigma}_2$=*diag*(10^{-4}, 10^{-4}), after a) 501; b) 515; c) 600; d) 1000 data records inserted.

Finding unusual trajectories in surveillance videos may lead to early identification of illegal behavior and hence the outliers need to be detected as soon as they appear. In such applications, due to data non-stationarity, static LOF algorithm is not suitable. To identify outliers in this environment, we have performed experiments on the second real life data set. The dataset corresponds to 239 video motion trajectories, where only 2 trajectories (225, 237) are visually identified as unusual behavior (person walking right and then back left and person walking very slowly). The trajectories were extracted from IR surveillance videos using our motion detection and tracking algorithms [29]. Each trajectory was represented by five equidistant points in [*x*,*y*,*time*] space (two spatial coordinates on the frame and the time instant) and the dimensionality of this feature vector was further reduced to three using the principal component analysis [30]. The dataset is available at www.cs.umn.edu/~aleks/inclof

Results of outlier detection on IR video trajectories are shown in Fig. 15. Fig. 15a shows computed values of *LOF*(*t*) using the incremental LOF algorithm, for each trajectory *t* at the moment of its insertion into the database. As it can be observed, the *LOF* values for trajectories 225 and 237 are significantly larger than for the rest of the trajectories, thus clearly indicating unusual behavior. To further illustrate performance of incremental LOF method, we plotted all considered trajectories in [*x*,*y*,*time*] space (Fig. 15b), color-coded by computed *LOF* value. It can be seen that the most intense color (and the highest *LOF*) corresponds to clear outliers.

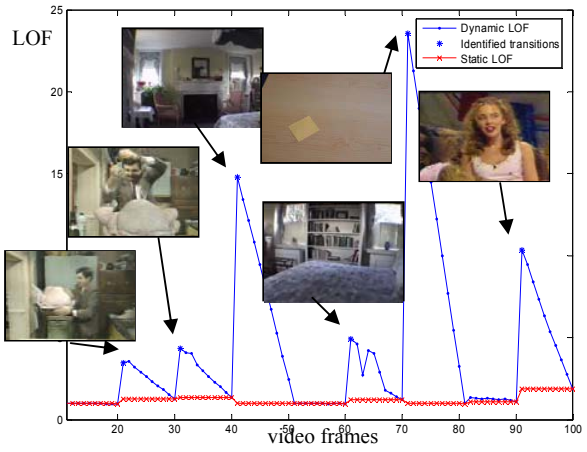


Fig. 14. Result of applying incremental (blue asterisks) and static "periodic" (red x) LOF algorithm (k=10) on video sequences from our test movie. For the static LOF algorithm, *LOF* values that are shown for each frame *t* are computed when the latest data record is inserted.

The third data set was 1998 DARPA Intrusion Detection Evaluation Data [31]. The original DARPA'98 data contains two types: training data and test data. The training data consists of seven weeks of network-based attacks inserted in the normal background data. Attacks in training data are labeled. The test data contain two weeks of network-based attacks and normal background data. Seven weeks of data resulted in about five million connection records. Since the amount of available data is huge (e.g. some days have several million connection records), we have chosen only the data from fifth week to present our findings. We used *tcptrace* utility software [32] as the packet filtering tool in order to extract information about packets from TCP connections and to construct new features. The DARPA'98 training data includes "list files" that identify the time stamps (start time and duration), service type, source IP address and source port, destination IP address and destination port, as well as the type of each attack. We used this information to map the connection records from list files to the connections obtained using *tcptrace* utility software and to correctly label each connection record as "normal" or an attack type. The same technique was used to construct KDDCup'99 data set [33], but this data set did not keep time information about the attacks. The main reason for this procedure is to associate new constructed features with the connection records from list files and to create more informative data set for learning. The complete list of the features extracted from "raw tcpdump" data using tcptrace software is available in our previously published results [4].

The results of applying the incremental LOF algorithm on DARPA98 data set are shown in Fig. 16. We have chosen to illustrate only two phenomena here. The first phenomenon corresponds to time moment t = 1368, when new behavior starts to appear. This behavior was clearly detected by incremental LOF algorithm (Fig 16, dash-dot cyan line). Although detected behavior did not correspond to any intrusions, it was important to include it when updating

characteristics of normal behavior. The second phenomenon at time moment t = 1937, corresponds to detection of Denial of Service (DoS) attack (Fig. 16, red dashed line). The *LOF* value at this time moment spikes very high, thus resulting in immediate DoS detection. If the static "periodic" LOF algorithm was applied after 2500 data records have been inserted into the data set, this DoS attack would not be detected (Fig 16, black dash-dot line), since it would create a new data distribution (scenario 1 (Fig.1) in section II).

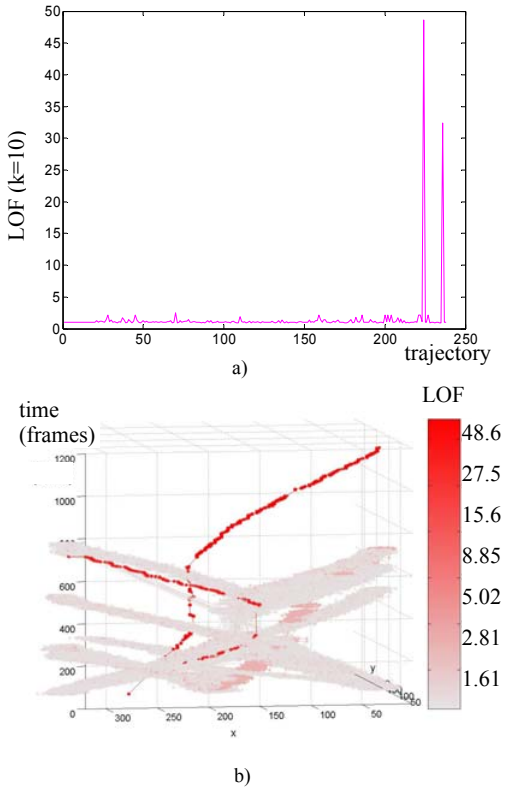


Fig. 15. Incremental LOF algorithm used for identifying unusual trajectories in motion videos (k=10); a) The values of *LOF* valuer obtained for each trajectory; b) Trajectories in [x,y,*time*] feature space colored according to the values of *LOF* value.

V. CONCLUSIONS AND FUTURE WORK

A framework for incremental density-based outlier detection scheme is presented. The proposed algorithm has the same detection performance as the static "iterated" LOF algorithm that is applied after insertion of each data record, but it is much more computationally efficient. Experimental results on several synthetic and real life data sets from video and intrusion detection domains indicate that the proposed incremental LOF algorithm can provide additional functionality that is difficult to achieve using static variants of LOF algorithm, including detection of new behavior as well as identification of masquerading outliers.

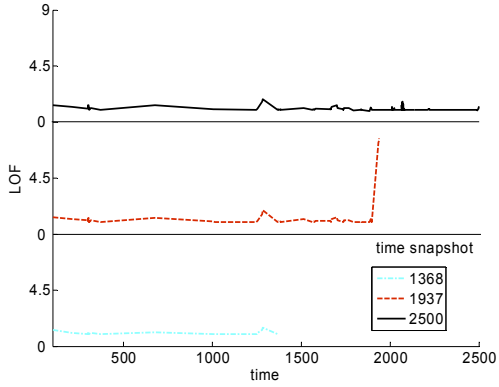


Fig. 16. Instant *LOF* values for all the data records in the dataset at time instants 1368, 1937, 2500 obtained using incremental LOF algorithm (k=10) on Network Intrusion dataset.

The fact that the number of updates in the incremental LOF algorithm per insertion/deletion of a single data record does not depend on the total number of data records is quite appealing for its use in real-time data stream applications. However, its performance crucially depends on efficient indexing structures to support k-nearest neighbor and reverse k-nearest neighbor queries. Due to limitations of existing indexing structures with high data dimensionality, the proposed incremental LOF (similar as static LOF) is not applicable when the data have large number of dimensions. The approximate k-NN and reverse k-NN algorithms might improve the applicability of incremental LOF with multidimensional data.

Future work on deleting data records from database is needed. More specifically, it would be interesting to design an algorithm with exponential decay of weights, where the most recent data records will have the highest influence on the local density estimation. In addition, an extension of the proposed methodology to create incremental versions of other emerging outlier detection algorithms, (e.g., Connectivity Outlier Factor (COF) [34], LOCI [35]), is also worth considering. Additional real-life data sets will be used to evaluate the proposed algorithm and ROC curves [38] will be applied to quantify the algorithm performance.

ACKNOWLEDGMENTS

D. Pokrajac has been partially supported by NIH (grant #2 P20 RR016472-04), DoD/DoA (award 45395-MA-ISP) and NSF (awards # 0320991, #HRD-0310163, #HRD-0630388). Authors would also like to thank Roland Miezianko, Terravic Corp. for providing IR surveillance video, Brian Tjaden, Wellesley College, for discussion about reverse k-NN queries and David Mount, University of Maryland, for insightful discussion about sphere covering verification.

REFERENCES

[1] M. Joshi, R. Agarwal, V. Kumar, PNrule, Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction, In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Santa Barbara, CA, May 2001.

[2] N. Chawla, A. Lazarevic, L. Hall, K. Bowyer, SMOTEBoost: Improving the Prediction of Minority Class in Boosting, In *Proceedings of the*

Principles of Knowledge Discovery in Databases, PKDD-2003, Cavtat, Croatia, September 2003.

[3] V. Barnett and T. Lewis, *Outliers in Statistical Data.* New York, NY, John Wiley and Sons, 1994.

[4] A. Lazarevic, L. Ertoz, A. Ozgur, J. Srivastava, V. Kumar, A comparative study of anomaly detection schemes in network intrusion detection, In *Proceedings of the Third SIAM International Conference on Data Mining*, San Francisco, CA, May 2003.

[5] A. Lazarevic, V. Kumar, Feature Bagging for Outlier Detection, In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* Chicago, IL, August 2005.

[6] N. Billor, A. Hadi and P. Velleman BACON: Blocked Adaptive Computationally-Efficient Outlier Nominators, *Computational Statistics & Data Analysis*, vol. 34, pp. 279-298, 2000.

[7] E. Eskin, Anomaly Detection over Noisy Data using Learned Probability Distributions, In *Proceedings of the International Conference on Machine Learning*, Stanford University, CA, June 2000.

[8] Aggarwal, C. C., Yu, P. Outlier detection for high dimensional data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

[9] M.M. Breunig, H.P. Kriegel, R.T. Ng and J. Sander, LOF: Identifying Density Based Local Outliers, In *Proceedings of the ACM SIGMOD Conference*, Dallas, TX, May 2000.

[10] E. Knorr and R. Ng, Algorithms for Mining Distance based Outliers in Large Data Sets, In *Proceedings of the Very Large Databases (VLDB) Conference*, New York City, NY, August 1998.

[11] D. Yu, G. Sheikholeslami and A. Zhang, FindOut: Finding Outliers in Very Large Datasets, *The Knowledge and Information Systems (KAIS) journal*, vol. 4, 4, October 2002.

[12] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo, A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data, in *Applications of Data Mining in Computer Security, Advances In Information* Security, S. Jajodia D. Barbara, Ed. Boston: Kluwer Academic Publishers, 2002.

[13] S. Hawkins, H. He, G. Williams and R. Baxter, Outlier Detection Using Replicator Neural Networks, In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery (DaWaK02)*, *Lecture Notes in Computer Science* 2454, Aix-en-Provence, France, pp. 170-180, September 2002.

[14] G. Medioni, I. Cohen, S. Hongeng, F. Bremond and R. Nevatia. Event Detection and Analysis from Video Streams, IEEE Trans. on Pattern Analysis and Machine Intelligence, 8(23), 873-889, 2001.

[15] Shu-Ching Chen, Mei-Ling Shyu, Chengcui Zhang, Jeff Strickrott: Multimedia Data Mining for Traffic Video Sequences. MDM/KDD 2001, pp 78-86.

[16] Shu-Ching Chen, Mei-Ling Shyu, Chengcui Zhang, Rangasami L. Kashyap: Video Scene Change Detection Method Using Unsupervised Segmentation And Object Tracking. *Proc. ICME* 2001

[17] Y. Tao, D. Papadias, X. Lian, Reverse kNN search in arbitrary dimensionality, In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, September 2004.

[18] Amit Singh, Hakan Ferhatosmanoglu, Ali Tosun, High Dimensional Reverse Nearest Neighbor Queries, In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'03)*, New Orleans, LA, November 2003.

[19] Ioana Stanoi, Divyakant Agrawal, Amr El Abbadi, Reverse Nearest Neighbor Queries for Dynamic Databases, *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dalas, TX, May 2000.

[20] J. Anderson, Brian Tjaden, The inverse nearest neighbor problem with astrophysical applications. In *Proceedings of the 12th Symposium of Discrete Algorithms (SODA)*, Washington, DC, January 2001.

[21] D. Pokrajac, L. J. Latecki, A, Lazarevic et al. *Computational geometry issues of reverse-k nearest neighbors queries,* Technical Report TR-CIS5001, Delaware State University 2006.

[22] J. Conway, N. H. Sloane, *Sphere Packings, Lattices and Groups*, Springer, 1998.

[23] F. P. Preparata, M. I. Shamos, *Computational Geometry: an Introduction*, 2nd Printing, Springer-Verlag 1988

[24] N. Roussopoulos, S. Kelley and F. Vincent, Nearest neighbor queries, 71-79, *Proceedings of the ACM SIGMOD Conference*, San Jose, CA, 1995

[25] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

[26] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for highdimensional data. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Databases*, pages 28–39, San Francisco, U.S.A., 1996. Morgan Kaufmann Publishers.

[27] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[28] D. DeMenthon, L. J. Latecki, A. Rosenfeld, and M. Vuilleumier Stückelberg: Relevance Ranking of Video Data using Hidden Markov Model Distances and Polygon Simplification. Proc. of the *Int. Conf. on Visual Information Systems*, Lyon, France, Springer-Verlag, pp. 49-61, November 2000.

[29] L. J. Latecki, R. Miezianko, V. Megalooikonomou, D. Pokrajac, Using Spatiotemporal Blocks to Reduce the Uncertainty in Detecting and Tracking Moving Objects in Video, *International Journal of Intelligent Systems Technologies and Applications,* 1 (3-4) 2006, pp. 376—392.

[30] I. T. Jolliffe. *Principal Component Analysis*, 2nd edition. Springer Verlag, 2002.

[31] R. P. Lippmann, D. J. Fried, I. Graf, J. et al, Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation, In *Proceedings DARPA Information Survivability Conference and Exposition (DISCEX)* 2000, Vol 2, pp. 12-26, IEEE Computer Society Press, Los Alamitos, CA, 2000.

[32] Tcptrace software tool, www.tcptrace.org.

[33] UCI KDD Archive, KDD Cup 1999 Data Set, www.ics.uci.edu/~kdd/databases/kddcup99/kddcup99.html

[34] J. Tang, Z. Chen, A. Fu, D. Cheung, Enhancing Effectiveness of Outlier Detections for Low Density Patterns, In *Proceedings of the Sixth Pacific-Asia Conference on Knowledge Discovery and Data Mining, (PAKDD)*, Taipei, May, 2002.

[35] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, C. Faloutsos: LOCI: Fast Outlier Detection Using the Local Correlation Integral, In *Proceedings of the 19th International Conference on Data Engineering* (*ICDE'03*), Bangalore, India, March 2003.

[36] S. D. Bay, M. Schwabacher, Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the Ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, 2003.

[37] D.M.V. Sommerville, *An Introduction to the Geometry of N Dimensions*, Dover Publications, Inc, NY, 1958.

[38] T. Fawcett, F. Provost, Activity monitoring: noticing interesting changes in behavior, Proceedings *fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, , August 15-18, 1999, San Diego, California, pp.53-62

[39] P. Domingos and G. Hulten, A General Framework for Mining Massive Data Streams, *Journal of Computational and Graphical Statistics* 12 (4) (2003), pp. 945-949.

[40] J.Tang, Z. Chen, AW. Fu, D.W. Cheung, Capabilities of outlier detection schemes in large datasets, framework and methodologies *Knowledge and Information Systems* 11(1), 2006, pp. 45–84.

[41] K. Yamanishi, J. Takeuchi, A unifying framework for detecting outliers and change points from non-stationary time series data. In *Proceedings of the Eighth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, July 23 - 26, 2002, pp. 676-681.

[42] E. Achtert E., C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, M. Renz, Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces.In *Proceedings ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'06)*, Chicago (IL), U.S.A., 2006, pp. 515-526.