Incremental Connectivity-Based Outlier Factor Algorithm

Dragoljub Pokrajac[1], Natasa Reljin[2], Nebojsa Pejcic[2], Aleksandar Lazarevic[3]

1 – CIS, AMTP and CREOSA, Delaware State University, 1200 North DuPont Highway, Dover, 19901 Delaware, USA
2 – AMTP, Delaware State University, 1200 North DuPont Highway, Dover, 19901 Delaware, USA
3 – UTRC and CREOSA, 411 Silver Lane, MS 129-15 East Hartford, CT 06108, USA
dpokrajac@desu.edu, natasa.reljin@gmail.com, npejcic08@desu.edu, lazarea@utrc.utc.com

Outlier detection has recently become an important problem in many industrial and financial applications. Often, outliers have to be detected from data streams that continuously arrive from data sources. Incremental outlier detection algorithms, aimed at detecting outliers as soon as they appear in a database, have recently become emerging research field. In this paper, we develop an incremental version of connectivity-based outlier factor (COF) algorithm and discuss its computational complexity. The proposed incremental COF algorithm has equivalent detection performance as the iterated static COF algorithm (applied after insertion of each data record), with significant reduction in computational time. The paper provides theoretical and experimental evidence that the number of updates per such insertion/deletion does not depend on the total number of points in the data set, which makes algorithm viable for very large dynamic datasets. Finally, we also illustrate an application of the proposed algorithm on motion detection in video surveillance applications.

Keywords Data mining, Data streams, Outlier detection, Incremental algorithm

1. 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]. 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 [2]. 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, flight record data) arrives continuously at an enormous pace thus posing a significant challenge to analyze it. 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 [3-5]. Automated identification of suspicious behavior and objects (e.g., people crossing the perimeter around protected areas, leaving unattended luggage at the airport, cars driving unusually slow or unusually fast or with unusual trajectories) based on information extracted from video streams is currently very active research area. In addition, there are many aviation systems that have incorporated Health and Usage Management systems as standard equipment on the aircraft to collect data during the flight. 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, there have been several density-based anomaly detection algorithms (LOF-Local Outlier Factor) algorithm [6], COF (Connectivity Outlier Factor) [7], LOCI (Local Correlation Integral) [8] that have been successfully applied in many domains for outlier detection in a batch mode. In this paper, we propose a novel incremental COF algorithm that is appropriate for detecting anomalies from data streams. There has only been a limited research in incremental (on-line) anomaly detection algorithms, and to the best of our knowledge, there is only incremental LOF algorithm [15] proposed so far. The proposed incremental COF algorithm provides equivalent detection performance as the static COF algorithm. The paper proves 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 both simulated and real life data sets have demonstrated that the proposed incremental COF algorithm can be very successful in detecting anomalies in various data streaming applications.

2. BACKGROUND

Outlier detection techniques [9] can be categorized into several groups: statistical, distance based, profiling and model-based approaches. In statistical techniques [e.g., 2], the data points are typically modelled by means of a stochastic distribution, and points are subsequently labelled as outliers with regards to their relationship with the model. Distance based approaches [e.g., 6, 8] detect outliers by using computed distances among points. Clustering-based techniques have been used to detect outliers either as side products of the clustering algorithms (e.g., points that do not belong to clusters or clusters that are significantly smaller than others) [e.g., 10]. In profiling methods, profiles of normal behaviour are built using different data mining techniques or heuristic-based approaches, and deviations from them are considered as outliers (approach used, e.g., in network intrusions). Model-based approaches first characterize the normal behaviour using some predictive models [e.g., 11], 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 [12, 13, 14] 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 [12]. In addition, Domingos and Hulten [13] proposed broad requirements that incremental algorithms need to meet, while Yamanishi and Takeuchi [14] used on-line discounting distributional learning of Gaussian mixture model and scoring based on the estimated probability density function. Pokrajac et al [15] proposed an incremental variant of local outlier factor algorithm, with asymptotic time complexity equal to the time complexity of static LOF algorithm.

In this study, we propose an incremental connectivity outlier factor (COF). The main idea of the static COF algorithm [7, 9] is to assign to each data record a degree of being outlier. This degree is called the *connectivity-based outlier factor (COF)* of a data record. Data records (points) with a high *COF* have average local connectivity smaller than their neighbourhood and typically represent strong outliers, unlike data records belonging to uniform clusters that usually tend to have lower *COF* values. The algorithm for computing the *COF*s for all data records has the following steps:

1. For each data record p find set $N_k(p)$ of its k nearest neighbours (k-*NN*);

2. Find set based nearest (*SBN*) path s and corresponding set based nearest trail (*SBN trail*) $e=\{e_1,\ldots e_k\}$ from p on $N_k(p)\cup\{p\}$. Here, the set based nearest (*SBN*) path from data record p_1 on set $N_k(p)$ is a sequence of records $s=\{p_1,p_2,\ldots,p_r\}$ such that for all $1 \le i \le r-1$, p_{i+1} is the nearest neighbour of set $\{p_1,\ldots,p_i\}$ in $\{p_{i+1},\ldots,p_r\}$ [7]. SBN trail is the sequence of edges $e = \{e_1,\ldots e_k\}$, where each edge connects two consecutive nearest neighbours from the SBN path.

3. Compute the average chaining distance from p_1 to $N_k - \{p_1\}$, denoted by $ac\text{-}dist_{Nk(p)\cup p}(p_1)$ and defined as:

$$ac-dist_{N_k(p)\cup p}(p) \equiv \sum_{i=1}^{k} \frac{2(k+1-i)}{k(k+1)} dist(e_i), \tag{1}$$

where $dist(e_i)$ denotes distance between nodes comprising an edge. The average chaining distance from p_1 to N_k-$\{p_1\}$ is the weighted sum of the cost description of the *SBN trail* for some SBN path from p_1, and can also be viewed as the average of the weighted distances in the cost description of the *SBN-trail*. Larger weights are assigned to the earlier terms [7].

4. Compute connectivity-based outlier factor (*COF*) at data record p with respect to its k-neighbourhood using the following formula:

$$COF(p) \equiv \frac{ac-dist_{N_k(p)\cup p}(p)}{\frac{1}{k}\sum_{o\in N_k(p)} ac-dist_{N_k(o)\cup o}(o)}. \tag{2}$$

COF is computed as ratio of the average chaining distance from data record p to $N_k(p)$ and the averaged average chaining distances at the record's neighbourhood.

In order to fully realize the need for *incremental* outlier detection techniques, it is important to understand that applying static COF outlier detection algorithms to data streams would be extremely computationally inefficient. Namely, static COF algorithm may be applied to data streams in the following "iterated" fashion: re-apply the static COF algorithm every time a new data record p_c is inserted into the data set. However, since every time a new record is inserted, the algorithm recomputes *COF* values for *all* the data records from the data set. With time complexity of COF algorithm of $O(n\cdot\log n)$ (for moderate dimension of the data records) [7], where n is the current number of data records in the data set, total time complexity for this "iterated" approach, after insertion of N records, is:

$$O\left(\sum_{n=1}^{N} n \log n\right) = O(N^2 \cdot \log N). \tag{3}$$

An alternative approach would be to perform static COF periodically after inserting particular data blocks. However, such "periodic" scheme cannot identify exact time when an outlier appears in the database and cannot detect outliers that would later be classified as normal records, due to non-stationarity in the dataset [15].

Our proposed incremental COF algorithm is designed to provably provide the same results in detecting outliers as the "iterated" COF. It is achieved by consistently maintaining for all existing records in the database the same *COF* values as the "iterated" COF algorithm. The incremental COF algorithm that we propose has time complexity $O(N \cdot \log N)$ thus clearly outperforming the static "iterated" COF approach. After all N data records are inserted into the data set, the final result of the incremental COF algorithm on N data records is independent of the order of insertion and equivalent to the static COF executed after all the data records are inserted.

3. METHODOLOGY

When designing incremental COF algorithm, we have been motivated by two aims. First, the result of the incremental algorithm must be equivalent to the result of the "iterated" static algorithm every time when a new record is inserted into a data set. Also, there should not be a difference between applying incremental COF and the static COF when all data records up to a considered time instant are available. Second, asymptotic time complexity of incremental COF algorithm has to be comparable to the static COF algorithm. In order to have feasible incremental algorithm, it is essential that, at any time moment, insertion/deletion of the data record results in small (preferably limited) number of updates of algorithm parameters. Specifically, the number of updates per insertion/deletion must be independent of the current number of records in the dataset. Otherwise, the time complexity of the incremental COF algorithm would be $\Omega(N^2)$ (N is the size of the final dataset). In this section, we provide efficient schemes for insertion and deletion of records for the incremental COF algorithm and discuss their time complexities.

3.1 Incremental COF algorithm

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

3.1.1. Insertion

In the insertion part of the incremental COF (incCOF) algorithm, the following two tasks are performed: a) insertion of new record in the database and computation of *ac-dist* and *COF* for the new record; b) maintenance, when *ac-dist* and *COF* values need to be updated for the records already existing in the database.

Consider an insertion of new record p into a database of two-dimensional records, see Fig. 1. According to Eq. (1), *ac-dist* may change for a certain record q if the set of its *k-nearest neighbours* $N_k(q)$ changes due to insertion of p. In other words, *ac-dist*(q) may change if a data record q is among *reverse k-nearest neighbours* of p. Since p is among *k-nearest neighbours* of q (see. Fig. 1a), *ac-dist(q)* should be updated. Set of records where *ac-dist* needs to be updated after insertion of p is denoted by $S_{update\_ac\_dist}(p)$ in the remaining part of the paper. Denote the set of records where *COF* should be updated as $S_{update\_COF}(p)$. According to Eq. (2), *COF*(o) for an existing record o needs to be updated if: a) *ac-dist*(o) is updated; b) the insertion of p changes the neighbourhood of o (in other words p is among *k-nearest neighbours* of o); c) *ac-dist* is updated for some of *k-nearest neighbours* of o.

Since *ac-dist*(*o*) is updated only if *o* is among reverse nearest neighbours of *p*, conditions a) and b) above would imply that $S_{update\_ac\_dist}(p) \subset S_{update\_COF}(p)$. Condition *c*) indicates that *COF* also needs to be updated for all reverse nearest neighbours of points from the set $S_{update\_ac\_dist}(p)$. For example, in Fig. 1a, *COF* will be updated on data record *r*, since its k-nearest neighbourhood contains a record from $S_{update\_ac\_dist}(p)$. The general framework for the insertion of new data record for the COF algorithm is given in Fig. 2a.

Similarly to the LOF approach [6], we define *k-th* nearest neighbour of a record *p* as a record *q* from the dataset *S* such that for at least *k* records $o' \in S - \{p\}$, it holds that $d(p,o') \leq d(p,q)$, and for at most *k*-1 records $o' \in S - \{p\}$, it holds that $d(p,o') < d(p,q)$. Here, *d(p,q)* denotes Euclidean distance between data records *p* and *q*. *k* nearest neighbours ($N_k(p)$) include all data records $r \in S - \{p\}$ such that $d(p,r) \leq d(p,q)$. We also define *k reverse nearest neighbours* of *p* (referred to as *kRNN* (*p*)) as all data records *q* for which *p* is among their *k* nearest neighbours. For a given data record *p*, $N_k(p)$ and *kRNN*(*p*) can be respectively retrieved by executing nearest-neighbour and reverse (a.k.a. inverse) nearest neighbour queries [16, 19] on a dataset *S*.

3.1.2. Deletion

In data stream applications, very often it may be necessary to delete irrelevant data records, due to their obsoleteness or change of regime. The general framework for deleting the block of data records S_d from the dataset *S* is given in Fig. 2b, and it is very similar to the insertion scheme. At the beginning, the record that needs to be deleted from the set is marked for deletion. Naturally, this removal of a record *p* may affect the *kRNN* of the other records. In addition, as in the insertion procedure, *ac-dist* of the *kRNN* of *p* is updated using Eq. (1). Due to update of *ac-dist*, the update of *COF* value also needs to be performed. This update, in addition to data records where *ac-dist* values change, also includes their reverse nearest neighbours. At the end, the data record can be deleted from the database.

We would like to discuss several technical aspects of the algorithm here. In the algorithms above, $S_{update\_ac\_dist}$ (*p*) is the set where *ac-dist* should be recomputed, not where the *ac-dist* would actually change. However, our experimental evidence suggests that *ac-dist* would *always* change for all points from $S_{update\_ac\_dist}$ (*p*). The second technical aspect is related to the question whether the reverse *k-nn* of points from $S_{update\_ac\_dist}$ should be computed with or without considering record *p* that is deleted. Here, we justify that during the deletion the data record *p* can be deleted from a database before S_{update_COF} is computed. Let *r* be an existing data record such that $r \notin S_{update\_ac\_dist}(p)$. Then, $p \notin N_k(r)$. Assume that $r \in S_{update\_COF}(p)$. Then, there exists $o \in S_{update\_ac\_dist}(p)$ such that $o \in N_k(r)$. Since $p \notin N_k(r)$, the deletion of *p* would not affect the neighbourhood of *r* and hence S_{update_COF} can safely be determined after *p* is deleted.

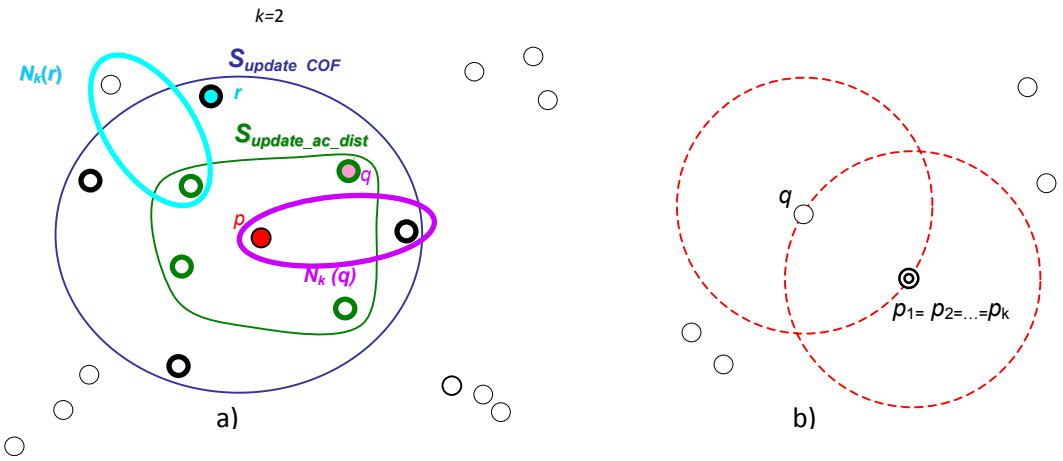
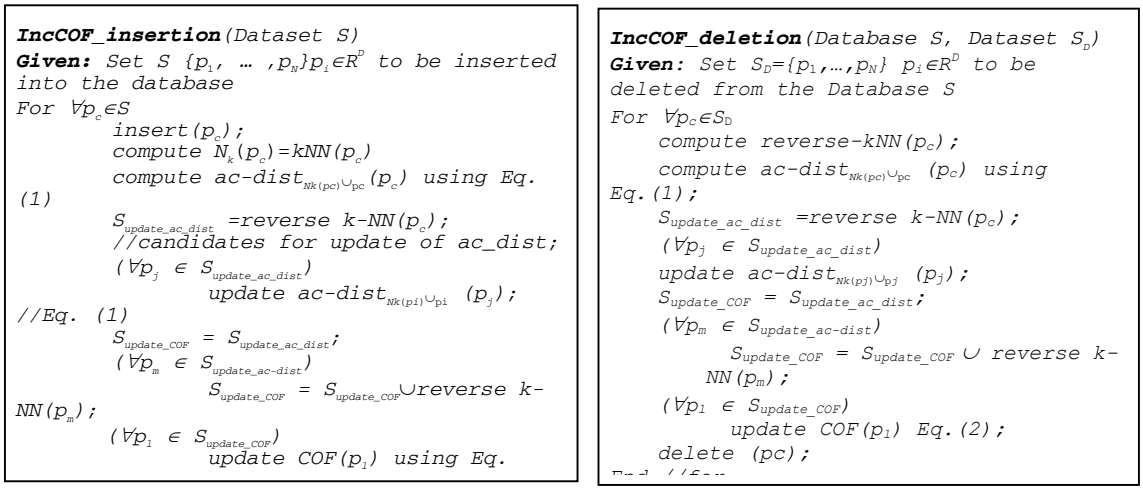


FIGURE 1: a) Illustration of updates for COF insertion; b) Degenerate case of incremental COF algorithm



(a) (b)

FIGURE 2: The pseudo code for a) insertion and b) deletion of incremental COF algorithm

The static COF algorithm [7,9] did not discuss a degenerate case when denominator in Eq. (2) is equal to zero. Such a scenario is possible since a database of dynamic data may contain identical records (corresponding to different time instants). Consider k identical records $p_1,\ldots,p_k$ and record q such that records $p_1,\ldots, p_{k+1}$ are k nearest neighbours of q and that q is among k nearest neighbours of each of records $p_1,\ldots,p_{k+1}$ (see Fig. 1b). In this case, it is easy to see that $ac-dist_{N_k(p_i)\cup p_i}(p_i)=0, i=1,...,k+1$ and that $ac-dist_{N_k(q)\cup q}(q)>0$. Hence, $COF(p_i)$ will involve division by 0, and therefore will be undefined. Recall, however, that *COF* is defined as the ratio of *ac-dist* at the data record and the average *ac-dist* at its neighbourhood. Hence, if the *ac-dist* at the record is equal to the average *ac-dist*, *COF* should be equal to 1. Therefore, for data records where *both* numerator and denominator of Eq. (2) are zeros, such as data records p_i, we define $COF(p_i)\equiv 1$. In contrast, when only denominator of Eq. (2) is equal to zero, such as for record q, the neighbourhood of the record will have infinitely larger density than the record itself. Hence, for such data records, we define $COF(q)\equiv\infty$ (in practice, we assign $COF(q)$ a very large number).

3.2 Computational efficiency of the incremental COF algorithm

In this section we provide a semi-formal overview of the computational complexity of the proposed algorithm. To efficiently implement incremental COF algorithm (*incCOF*), similarly as in static COF algorithm [7], we may use an intermediate database *M* in which for every data record there is a list of its *k*-nearest neighbours along with their distances to that record, *SBN* trail and its cost description.

During the insertion step of incCOF, after a new record p_c is inserted into a database, its *ac-dist* needs to be computed. This requires execution of nearest neighbour (*k*-nn) query and insertion of a new record into the intermediate database *M*. Afterwards, we execute *kRNN* query and update $|S_{update\_ac\_dist}|$ data records in database *M* corresponding to reverse nearest neighbours p_j of record p_c (during this step, we can "*on-fly*" compute updated values of *ac-dist*). Then, for each record p_j we need to execute another *kRNN* query to determine S_{update_COF}, which includes $|S_{update\_COF}|$ updates of the intermediate database. Finally, we compute *COF* for all records in S_{update_COF} and for the inserted record p_c. Hence, when inserting a new record, there is a need to perform one kNN query, $1+|S_{update\_ac\_dist}|$ kRNN queries, $O(|S_{update\_COF}|+|S_{update\_ac\_dist}|)$ updates of the database *M* (including computation of *ac-dist*), and $1+S_{update\_COF}$ computations of *COF*. The computational complexity for deletion is similar to complexity of insertion procedure. When deleting a data record, kRNN query needs to be executed to determine records for which *ac-dist* will be updated. For each such record, another *kRNN* is executed to determine set of records

where *COF* need be recomputed. During each calculation of *kRNN*, data records in database *M* corresponding to discovered reverse nearest neighbours p_j need to be updated. Hence, for deletion phase of incCOF, there is a need to perform $1+|S_{update\_ac\_dist}|$ kRNN queries, $O(|S_{update\_COF}|+|S_{update\_ac\_dist}|)$ updates of the database *M* (including computation of *ac-dist*), and S_{update_COF} computations of *COF*.

When efficient algorithms for kNN [e.g. 20], kRNN [e.g., 16-18], as well as efficient indexing structures for inserting / deleting data records [21,22] are employed, the time complexity of kNN and kRNN search algorithms is logarithmic in the current size *N* of the database. Time needed to perform operations involving a record of intermediate database *M* does not depend on the total number of records *N* [7]. Hence, to determine asymptotic time complexity and feasibility of the proposed incremental COF algorithm, it is essential to demonstrate that the number of affected data records (updates of *ac-dist* and *COF values*) does not depend on the current number *N* of records in the dataset.

Let $f(p_c)$ be the number of reverse nearest neighbours p_j of the record p_c. Then, the number of *ac-dist updates* is equal to $|S_{update\_ac\_dist}|= f(p_c)$ while the total number of *COF* updates is equal to $|S_{update\_COF}|= f(p_c)+\sum_{p_j \in kRNN(p_c)} f(p_j)$. It has been shown [23] that $\sup f(.) \equiv F = \Theta\left(k2^D\sqrt{D}\right)$ where *D* is the dimensionality of the dataset. In addition it is explicitly demonstrated that *F* does not asymptotically change with respect to the number of records currently in the database. Therefore, $|S_{update\_ac\_dist}| = O\left(k2^D\sqrt{D}\right)$, $|S_{update\_COF}|= O\left(k^2D2^{2D}\right)$. As a consequence, the computational complexity of one insertion/deletion step of the proposed algorithm is $O\left(k^2D2^{2D}+k2^D\sqrt{D}\log(N)\right)$. For a sequence of *N* insertions/deletions on an empty database, the computational complexity then becomes $O(N\log(N))$.

Aforementioned discussion considers worst-case theoretical performance of the algorithm. Practical computational time per insertion/deletion depends on the position of the inserted/deleted record with respect to other records in the database (here referred to as "boundary effect") as well as on the actual number of records (for small enough *N*).

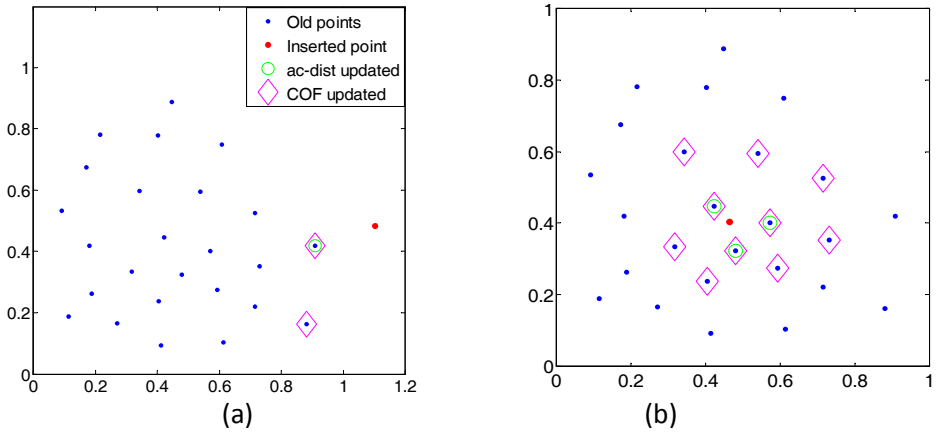


FIGURE 3: a) Insertion of a point on the boundary of the cluster: 1 *ac-dist* updated, 2 *COF* values updated (*k*=3); b) Insertion of a point inside the cluster: 3 *ac-dist* updated, 10 *COF* values updated (*k*=3)

Fig. 3 illustrates previously mentioned "boundary effect". The experiments from Fig. 3 are performed on data uniformly distributed within a unit circle. During the COF insertion (or deletion), the number of updates may depend on the position of the inserted/deleted data record with respect to to the existing data clusters. If a record is on the boundary of a cluster, or far from any cluster, it is possible that the record will not have any reverse nearest neighbours. In that case, the number of records where *ac-dist* (and hence *COF*) is updated will be zero. Fig. 3a shows a case of incCOF insertion (*k*=3), where a data record that will be inserted lies on the boundary of a cluster. The record has only one reverse *k*-nearest neighbour, so *ac-dist* will be

updated for only one data record. Consequently, *COF* is updated at only two records. In contrast, when a record is inserted inside the existing cluster, the number of reverse neighbours increases. Fig. 3b contains an example where *ac-dist* and *COF* are updated at 3 and 10 points, respectively.

Fig. 4 further illustrates the boundary effect and the influence of the number of records *N* in the database on the number of updates for *N* that is not sufficiently large. When the database contains *N*=200 records, insertion of the new record causes the updates that for *k*=20 occur approximately within a circle with a radius of 0.6. This circle cannot spread across the boundary of the cluster, which ultimately limits the number of data records where *COF* is updated. Scenario when *N*=5000 is much different. The updated records are confined in much smaller area which is far from the boundary of the cluster. Hence, when *N* is large enough, the boundary effect will have less influence on the selection of records where the updates occur, and the number of *COF* and *ac-dist* updates would stabilize. For deletion of the data record, the results would be the same (e.g., if we decide the record denoted by a pentagram on Fig. 4, the same records would be affected as when we inserted the record).

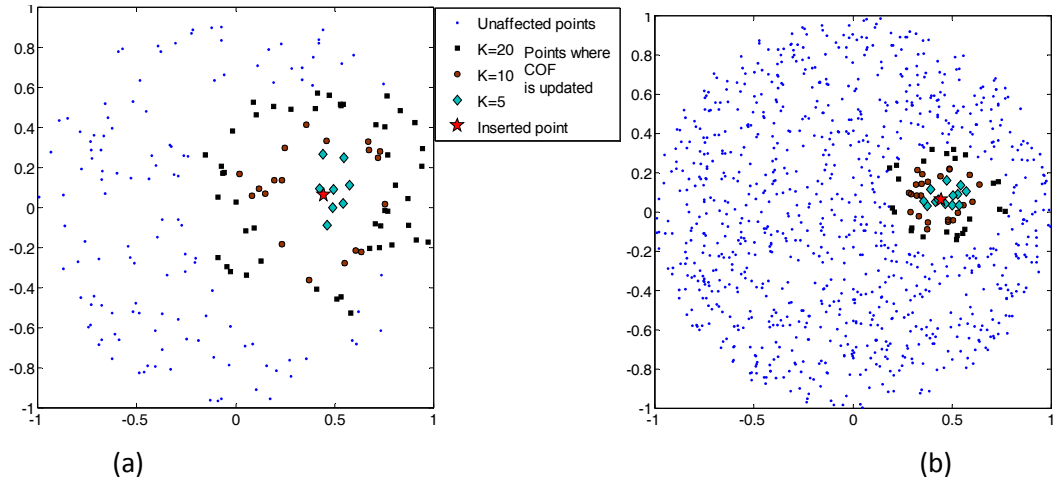


FIGURE 4: Points where *COF* is updated when a point is inserted into a cluster with *N* points ((a) ***N***=200, b) ***N***=5000) uniformly distributed within a random circle, for different values of *k*

4. EXPERIMENTAL RESULTS

4.1 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 *COF* values and the total number of data points *N*; (ii) the dependence of the number of updates for *COF* and *ac-dist* values on *COF* parameter *k*; and (iii) the dependence of the number of updates for *COF* values on the dimension *D*. The synthetic data sets had different number of data records ($N \in \{100,200,\ldots,5000\}$), as well as different number of dimensions ($D \in \{2,3,4,5,6,7,8,9,10\}$). For each pair (*D, N*), we have created 10 data sets with *N* random records generated from *D*-variate distribution. We experimented with uniform and standard (zero mean, unit covariance matrix) Gaussian distribution, and with distribution uniform within a unit hypersphere. For each of 10 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 *ac-dist*, *COF* values in the incremental COF algorithm for insertion of a new data record into the dataset. In this study we report results when the database consisted of records uniformly distributed with a unit hypersphere, where new data record is inserted randomly (with a uniform distribution) within a hypersphere with center at origin and radius 0.5. This way, we tried to reduce boundary effect discussed in the previous session. Results obtained for the data sets generated using other considered distributions are analogue and not reported here due to lack of space.

First, we wanted to demonstrate theoretical result that asymptotically the number of *COF* updates does not depend on the number of points in the database. We performed three-way ANOVA analysis (factors *N*,*D* and *k*), and then followed-up with Tukey multiple comparison. The following Fig. 5 shows estimated means and confidence intervals for the number of *COF* updates, for each pair of *N* and *k*. Analyzing this figure, it can be clearly observed that for sufficiently large *N*, the number of updates ceases to be significantly dependent on *k*. The explanation is easy: for sufficiently large *N*, the boundary effect becomes negligible (see Fig. 4b) and hence the number of updates stabilizes. From this figure, it is also clear that *N* for which the number of updates stabilizes increases with larger *k*.

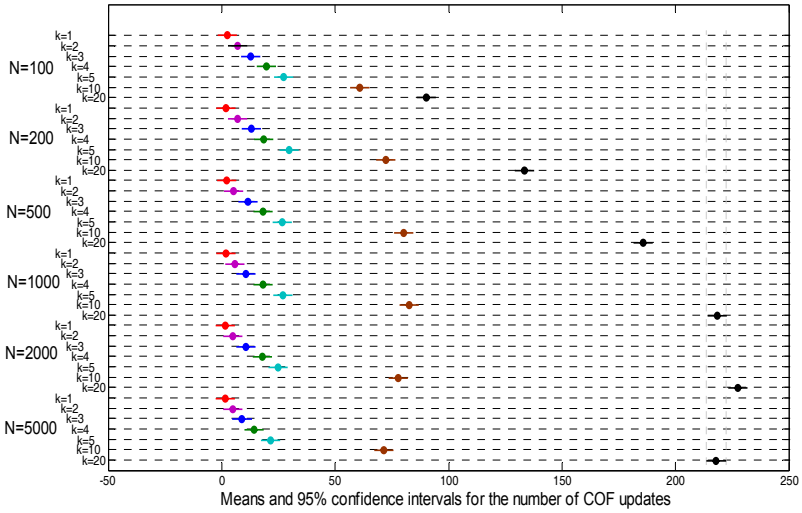


FIGURE 5: Multiple comparison analysis for the number of COF insertions into simulated dataset

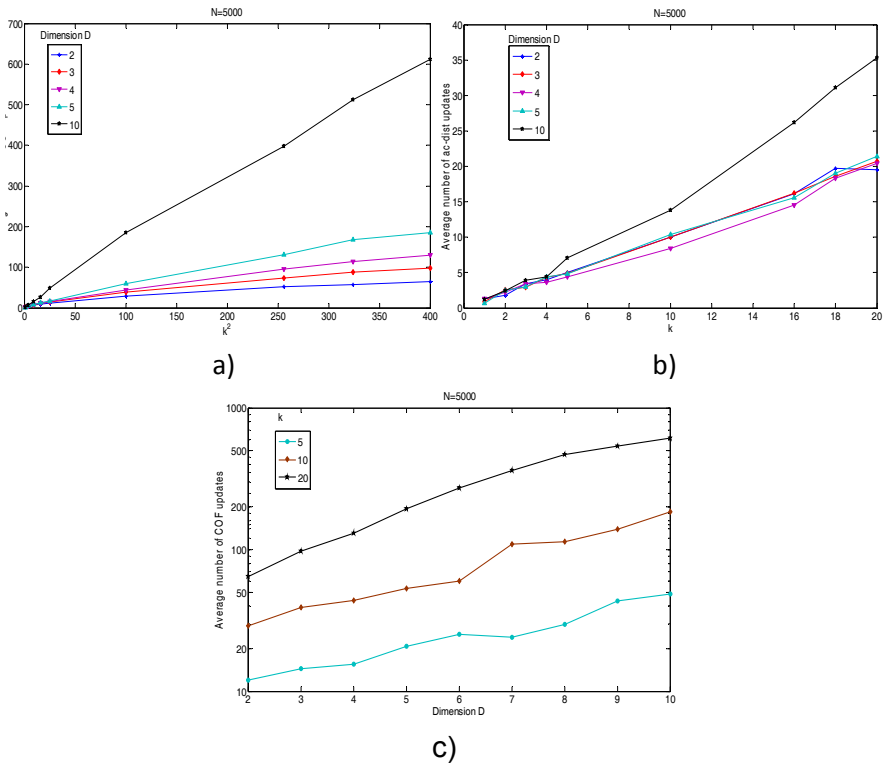


FIGURE 6: a) Average number of COF updates for different dimensionalities *D*, as a function of k^2; b) Average number of *ac-dist* updates for different dimensionalities *D*, as a function of *k*; c) Dependence of average number of COF updates on dimensionality

In section 3, we have shown that theoretically the number of *COF* and *ac-dist* updates grows proportionally to k^2 and to k, respectively. To check this, we plotted average number of *COF* and *ac-dist* updates for various values of dimensionality D, and for N=5000 (where the number of updates stabilize). Fig. 6a shows number of *COF* updates as function of k^2. We can see that *in fact*, the number of updates grows slightly less than k^2. This could be partly attributed to the boundary effect. Fig. 6b shows number of *ac-dist* updates as function of k. This figure agrees with a theoretical postulate that the worst-case number of *ac-dist* updates is proportional to k.

Fig. 6c shows the dependency of the average number of COF updates on data dimension, for three practically relevant values of k. Since the figure is semilog, it verifies theoretical finding that the number of COF updates is exponentially dependent on dimension. Observe that there is some "saturation" for larger dimensionalities and k=20, that can be attributed to boundary effect that is more prevalent for large dimensionality, and larger k. Similar behaviour can be observed for *ac-dist* (figures omitted due to space limitations).

4.2 Experiments on real life data sets

To demonstrate effectiveness of the proposed incremental COF algorithm on a real-life dataset, we employed incCOF for motion detection in surveillance videos. Here we present results on a video sequence from the Performance Evaluation of Tracking and Surveillance (PETS) repository[1]. To construct features for outlier detection, we split each frame from a video sequence into disjoint blocks containing 8×8 pixels and obtained spatiotemporal (3D) blocks by combining spatial blocks from the three consecutive frames at the same video plane location. The dimensionality of obtained 3D blocks (which are initially represented as 192 dimensional vectors of gray level pixel values) is reduced into three using linear principal component analysis (PCA). The resulting dataset contained $36\times48\times2482$ three-dimensional vectors, each representing one spatial-temporal block at one of 2482 time instants. More details about data preprocessing can be found in [24].

We used vectors from each of 36×48 block locations as input of incCOF algorithm. Similar as in [18], we define a *motion orbit* as path traversed in time by the vector of the PCA components representing a block at a specific location. As demonstrated in [18], vectors corresponding to background frames at the observed block location tend to group in dense clusters. In contrast, vectors corresponding to motion correspond to the elongated 1-dimensional orbits. IncCOF algorithm is demonstrated useful in performing motion detection in this setting. Fig. 7a shows motion orbit corresponding to block location (24,28). In the first 490 frames, there was no motion at the block location, corresponding to the "black" cluster in the Fig. 7a. Significant motion occurs, e.g., between frames 820 and 870, which corresponds to red-labeled elongated orbit. Fig. 7b shows results of applying incCOF on the vectors corresponding to the same block. We varied parameter k of the algorithm. For each vector, we plot the value of *COF* computed after the insertion of the vector into the database. Regions of low *COF* factor correspond to frames where there was no motion at the observed spatial block (e.g., frames 1-490). In regions where there is motion at the observed block, *COF* had high values (e.g., frames 820-870, see inset of Fig. 7b). The crisp identification of motion regions can be obtained by thresholding of computed *COF*, and is subject of a forthcoming manuscript. The best results were obtained using k=20. For small values of k, computed *COF* factor tends to fluctuate, and has steep and narrow peeks. Hence, the motion identified by using low k would typically be shaky and zig-zag. The results of motion detection by thresholding the *COF* factor can be seen at http://tesla.cis.desu.edu/COF (the choice of threshold is out of scope of this paper).

[1] ftp://pets.rdg.ac.uk/PETS2001/DATASET1/TESTING/CAMERA1_JPEGS/

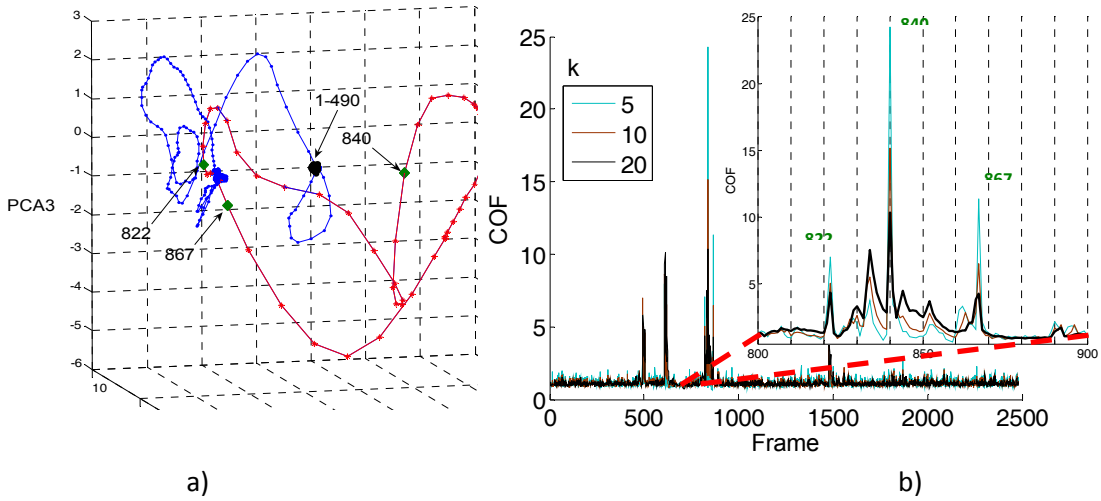


FIGURE 7: a) "Motion orbit" of vectors representing spatial-temporal blocks at location (24,28); b) Instantaneous values of *COF* computed for block (24,28) for three different values of *k* (5,10,20)

5. CONCLUSIONS AND FUTURE WORK

A framework for incremental outlier detection scheme is presented. The proposed algorithm has the same detection performance as the static "iterated" COF algorithm that is applied after insertion of each data record, but it is significantly more computationally efficient. Experimental results on synthetic and real life data sets from video surveillance domain indicate that the proposed incremental COF algorithm can provide additional functionality that is difficult to achieve using static variants of COF algorithm, including detection of new behaviour.

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

Future work on deleting data records from database is needed. E.g., 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. Additional real-life data sets will be used to evaluate the proposed algorithm and ROC curves [12] will be applied to quantify the algorithm performance. Specifically, we plan to apply incremental COF algorithm in network intrusion detection, detection of anomalous trajectories in video surveillance, etc. Performance comparison of incremental COF and LOF is also needed. Also, we currently work on development of parallel/distributed version of the incremental COF algorithm.

ACKNOWLEDGEMENTS

D. Pokrajac, N. Reljin and N. Pejcic have 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 Brian Tjaden, Wellesley College, for discussion about reverse k-NN queries and David Mount, University of Maryland.

REFERENCES.

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

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

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

[4] Chen, S.-C., Shyu, M.-L., Zhang, C. and Strickrott, J. (2001) Multimedia Data Mining for Traffic Video Sequences. *MDM/KDD*, pp 78-86.

[5] Chen, S.-C., Shyu, M.-L., Zhang, C. and Kashyap, R.L. (2001) Video Scene Change Detection Method Using Unsupervised Segmentation And Object Tracking. *Proc. ICME.*

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

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

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

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

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

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

[12] Fawcett, T., Provost, F. (1999) Activity monitoring: noticing interesting changes in behavior. *Proceedings Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, pp. 53-62, 15-18 August.

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

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

[15] Pokrajac, D., Lazarevic, A. and Latecki, L.J. (2007) Incremental Local Outlier Detection. *Proc. CIDM*, April.

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

[17] Singh, A., Ferhatosmanoglu, H. and Tosun, A. (2003) High Dimensional Reverse Nearest Neighbour Queries. *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM'03)*, New Orleans, LA, November.

[18] Stanoi, I., Agrawal, D. and Abbadi, A.E. (2000) Reverse Nearest Neighbour Queries for Dynamic Databases. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Dalas, TX, May.

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

[20] Roussopoulos, N., Kelley, S., and Vincent, F. (1995) Nearest neighbour queries. *Proceedings of the ACM SIGMOD Conference*, San Jose, CA, pp. 71-79.

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

[22] Berchtold, S., Keim, D. Kriegel, H.-P. (1996) The X-tree: An index structure for high dimensional data. *Proceedings 22nd International Conference on Very Large Databases*, San Francisco, CA, pp. 28–39,.

[23] Pokrajac, D., Petkovic, M., Latecki, L.J., Lazarevic, A., Reljin, N. and Milutinovic, J. (2008) Computational Geometry Issues of Reverse Nearest Neighbour Algorithm. *Proc. 7th Annual Hawaii International Conference on Statistics, Mathematics and Related Fields,* Hawaii, January.

[24] Latecki, L.J., Miezianko, R., Megalooikonomou, V. and Pokrajac, D. (2006) 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)**, pp. 376—392.