Efficient and Flexible Algorithms for Monitoring Distance-Based Outliers over Data Streams

Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, Yannis Manolopoulos*

Department of Informatics, Aristotle University

54124 Thessaloniki, Greece

{*kontaki,gounaria,papadopo,tsichlas,manolopo*}*@csd.auth.gr*

Abstract

Anomaly detection is considered an important data mining task, aiming at the discovery of elements (known as outliers) that show significant diversion from the expected case. More specifically, given a set of objects the problem is to return the suspicious objects that deviate significantly from the typical behavior. As in the case of clustering, the application of different criteria leads to different definitions for an outlier. In this work, we focus on distance-based outliers: an object x is an outlier if there are less than k objects lying at distance at most R from x. The problem offers significant challenges when a stream-based environment is considered, where data arrive continuously and outliers must be detected on-the-fly. There are a few research works studying the problem of continuous outlier detection. However, none of these proposals meets the requirements of modern stream-based applications for the following reasons: i) they demand a significant storage overhead, ii) their efficiency is limited and iii) they lack flexibility in the sense that they assume a single configuration of the k and R parameters. In this work, we propose new algorithms for continuous outlier monitoring in data streams, based on sliding windows. Our techniques are able to reduce the required storage overhead, are more efficient than previously proposed techniques and offer significant flexibility with regards to the

*Corresponding author: A. Gounaris, gounaria@csd.auth.gr

input parameters. Experiments performed on real-life and synthetic data sets verify our theoretical study.

---

1. Introduction

Anomaly detection is a data mining task focusing on the discovery of objects, called *outliers*, that do not seem to have the characteristics of the general population. To quote Johnson [1]: "*an outlier is an observation in a data set which appears to be inconsistent with the remainder of that set of data*". For example, from a statistical point of view, an object is an outlier if it deviates significantly from the distribution.

Outlier discovery is performed for two main reasons: i) removing the outliers before executing a clustering task leads to more effective cluster formation and ii) outliers may not always be noise, but they may represent interesting elements that diserve further exploration (e.g., a large beautiful house sold in a very low price). Thus, either being noise or useful information, outliers should be mined efficiently.

One of the most widely used outlier definitions is the one based on distance: an object x is considered as an outlier, if there are less than k objects in a distance at most R from x, excluding x itself. On the other hand, if the number of objects in the R-neighborhood of x are enough (i.e., more than k), then x is characterized as an *inlier*. The outliers defined this way are termed *distance-based* outliers [2, 3], and the corresponding type of outlier detection has the advantages of detailed granularity of analysis and detecting isolated groups of outliers [6]. Note that, to characterize an object $x \in U$ as an outlier or an inlier, we just need a way to compute the distance between x and any other object of the universe U. If the objects are represented as points in a multi-dimensional space, then the distance can be any L_p norm (e.g., Euclidean or Manhattan). However, many applications require distance computations based on more expensive distance measures such as the Jaccard distance for near duplicate detection, the edit distance for sequence alignment in bioinformatics,

distances based on quadratic form in multimedia applications and many more. Therefore, it is meaningful to provide general techniques that can work using many different distance measures and not to focus solely on multi-dimensional spaces.

Another important issue affecting the way outliers are mined is the dynamic nature of the universe U under consideration. In a static data set, we do not expect any changes in the outliers, since there are no insertions, deletions or updates. However, such data sets are rare in modern applications, which on the contrary require data mining tasks were changes are very frequent. Thus, in this work we focus on *data streams*, where the contents of U change continuously and, consequently, the set of outliers must be updated accordingly.

In data stream applications, data volumes are huge, meaning that it is not possible to keep all data memory resident. Instead, a *sliding window* is used, keeping a percentage of the data set in memory. The data objects maintained by the sliding window are termed *active objects*. When an object leaves the window we say that the object *expires*, and it is deleted from the set of active objects. There are two basic types of sliding windows: i) the *count-based window* which always maintains the n most recent objects and ii) the *time-based window* which maintains all objects arrived the last t time instances. In both cases, the *expiration time* of each seen object is known. The challenge is to design efficient algorithms for outlier monitoring, considering the expiration time of objects. Another important factor of stream-based algorithms is the memory space required for auxiliary information. Storage consumption must be kept low, enabling the possible enlargement of the sliding window, to accommodate more objects.

Contributions. In this work, we design efficient algorithms for continuous monitoring of distance-based outliers, in sliding windows over data streams, aiming at the elimination of the limitations of previously proposed algorithms. Our primary concerns are efficiency improvement and storage consumption reduction. A secondary concern stems from the problem of effective parameter

configuration that application developers face; more specifically, it is hard to set R and k a-priori in a way that meets the user needs. To address this, we allow for multiple configurations to be set and evaluated concurrently thus improving on the algorithm flexibility. In summary, our main contributions are the following:

- We prove a linear space lower bound which implies that in order to answer outlier queries on a set of objects one needs to store information about all objects even if we are settled with an approximate answer with a probability of success. This means that the window size W fully determines the number of stored objects, which are $O(W)$. This is a serious setback since in the various streaming models (e.g., [4]) we always strive for efficiency in queries as well as (asymptotic) minimization of space in order to support queries on larger sets of data. This is because the size W of the sliding window silently determines the size of the memory as well as the"interesting objects" to consider.

- A novel continuous algorithm is designed, which has two versions, and requires the radius R to be fixed but can handle multiple values of k. This algorithm (COD) consumes significantly less storage than previously proposed techniques and in addition, is more efficient.

- Since different users may have different views of outliers, we propose a new algorithm (ACOD) able to handle multiple values of k and multiple values of R, enabling the concurrent execution of different monitoring strategies.

- We propose an algorithm (MCOD) based on micro-clusters [5], to reduce the number of distance computations. There are cases where the distance function used is very expensive, and therefore, there is a need to keep this number low. This algorithm is also extended to support multiple queries (AMCOD).

- Performance evaluation results are offered based on real-life as well as synthetically generated data sets. The results show that our algorithms

are consistently more efficient.

Roadmap. The rest of the work is organized as follows. Section 2 discusses related work in the area, whereas Section 3 presents some important preliminary concepts, to keep the article self-contained. In Section 4 we prove a lower bound on the space required to solve the problem. This bound essentially says that that in order to monitor outliers in one pass we need to store information about all objects. We present our techniques in Section 5, whereas Section 6 contains the performance evaluation results based on real-life and synthetic data sets. Finally, Section 7 concludes the work and briefly discusses future work in the area.

2. Related Work

Outlier detection is a topic that has been attracting the interest of researchers for several decades. Comprehensive surveys can be found in [6, 7, 8, 9, 10]. We distinguish between two main categories of techniques, static and steaming ones, which are discussed in turn.

2.1. Static Outlier Detection

Most of the early techniques originate from the statistics community [1, 11], where the objects are modeled as a distribution, and objects are marked as outliers depending on their deviation from this distribution. However, for large dimensionalities and complex data types, statistical techniques fail to model the distribution accurately, leading to performance degradation. In addition, these techniques do not scale well for large databases.

The problem of outlier detection has been also addressed by the database and data mining communities, aiming at solving the problem of scalability to large datasets mentioned above, shifting the focus on tailored data structures and adaptations to specific environments, such as sensors, time series, texts, and so on. Outlier detection has been studied both in the context of multi-dimensional data sets [12] and in the more general case of metric spaces [13]. Usually, the proximity among objects is used to decide if an object is an outlier

or not. However, specialized techniques may also be applied (e.g., projections in the case of multi-dimensional data). In [14], the local reachability density is used to mark an object as an outlier. Distance-based outliers, considered in our work, follow a proximity-based approach and employ another simple and intuitive definition [2, 3], where an object is considered an outlier if there is a limited number of objects in its neighborhood, or equivalently, if the distance of a data point to its k-nearest neighbor exceeds a threshold [6]. Additional outlier definitions exist; for example, graph generation is used in [15] in order to model geometrical structure and exploit local graph connection for subspace outlier detection. Outlier elimination from aggregate query results by discovering appropriate predicates is proposed in [16]. Recent trends in outlier detection include the investigation of techniques that are particularly significant for big data analysis, such as dealing with high dimensionality, e.g., [17, 18], and considering graph-based data types common in social networks, e.g., [19, 20], categorical data [21] and video streams [22]. Apart from the fact that outliers are important in many applications, their discovery allows the data set to be "cleaned" to apply a particular model [23], while in many cases, their detection is a by-product of clustering, e.g., as in [24].

2.2. Streaming Outlier Detection

The fundamental characteristic of the majority of the proposed algorithms is that they operate in a static fashion. This means that the algorithm must be executed from scratch if there are changes in the underlying data objects, leading to performance degradation when updates are frequent. A special case with extremely high interest is the streaming case, where objects arrive in a streaming fashion [25], and usually in high rates. In this case, traditional algorithms fail to meet the processing requirements and therefore, specialized stream-based techniques emerge. One of the data mining tasks studied under the streaming model is clustering, where we are interested in clustering either a single stream or multiple streams. Similarly, anomaly detection over data streams is another emerging task with many applications like real-time fraud/spam detection, com-

puter network abuse, stock monitoring.

Among the various streaming techniques, we focus on sliding window methods, which have been used extensively. Since the stream is continuously updated with fresh data, it is impossible to maintain all of them in main memory. Therefore, a window is used which keeps track of the most recent data and all mining tasks are performed based on what is "visible" through the window. As reported in [7], most window-based models are currently offline. The most relevant research works are [26] and [27] which both consider the problem of continuous outlier detection in window-based data streams, without limiting their techniques to multi-dimensional data. However, both methods have some serious limitations that are tackled in this work.

In this research, we improve upon the two proposals mentioned above and propose four algorithms for continuous outlier monitoring over data streams. In comparison to existing approaches our techniques manage to reduce the running time and the storage requirements. In addition, our techniques offer significant flexibility regarding the parameter values, enabling the execution of multiple distance-based outlier detection tasks with different values of k and R. Moreover, by using the concept of micro-clusters, we manage to reduce the number of distance computations. This work is an extension of the research carried out in [28]; the main extensions include the theoretical analysis, the proposal of the AMCOD variant and additional evaluation (e.g., for arbitrary window slides).

Other proposals that deal with the broader problem of outlier detection in data streams include detection of changes, e.g., [29]; consideration of discrete sequences, e.g., [30]; techniques that rely on estimating the deviation from the expected values in time-series, e.g., [31] and density, e.g., [32]; specialized techniques for sensor networks, e.g., [33], and probabilistic streams, e.g., [34, 35]; and solutions for the high-dimensionality problem in streaming outlier detection, e.g., [36]. Distance-based outlier detection has been also considered in [37] without considering incremental outlier computation though, [38], which employs data editing techniques, and [39], which focuses on efficient correlation computation techniques for multiple time series. Finally, in [40], an anytime

technique is presented, which adopts a best-effort approach given the available time to process each element in the stream.

3. Fundamental Concepts

This section serves a two-fold purpose: firstly to formalize the problem, and secondly to explain in more depth the rationale and the limitations of existing approaches. Table 1 summarizes the most frequently used symbols throughout the article, along with their interpretation.

Sliding window semantics can be either time-based or count-based. In time-based window scenarios, the window size W and the $Slide$ are both time intervals. Each window has a starting time T_{start} and an ending time $T_{end} = T_{start} + W$. The window slide is triggered periodically by the system time (wall clock time), causing T_{start} and T_{end} to increase by $Slide$. Each window contains

Table 1: Frequently used symbols.

Symbol	Interpretation
q_i	the i-th query
$\mathcal{Q}$	the set of queries
W	the window size; $q.W$ is the size of the window for query q
$Slide$	the window slide
$\mathcal{P}$	the set of objects in the current window (active objects)
n	the number of non-expired objects ($n = \|\mathcal{P}\|$)
p_i	the i-th object, $i = 1, ..., n$
$p_i.arr$	the arrival time of object p_i
$p_i.exp$	the expiration time of object p_i
now	the current time instance
R	the distance parameter for the outlier detection; $q.R$ is the distance parameter for query q
k	the number of neighbors parameter; $q.k$ is the neighbors' parameter for query q
$\mathcal{I}(R,k)$	the set of inliers (i.e., non-outliers) for specific R and k
$\mathcal{D}(R,k)$	the set of outliers for specific R and k
S_{p_i}	the set of succeeding neighbors of p_i
$n^+_{p_i}$	the number of succeeding neighbors of p_i ($n^+_{p_i} = \|S_{p_i}\|$)
P_{p_i}	the set of preceding neighbors of p_i
$n^-_{p_i}$	the number of preceding neighbors of p_i ($n^-_{p_i} = \|P_{p_i}\|$)
nn_{p_i}	the total number of neighbors of p_i, $nn_{p_i} = n^+_{p_i} + n^-_{p_i}$

a set $\mathcal{P}$ of n objects. In general, n varies between sliding windows reflecting the differences in arrival rates. The non-expired objects are those whose arrival time $p.arr \geq T_{start}$. An object expires after x slides, where $x = \lceil \frac{W}{Slide} \rceil$; $p.exp$ is the expiration time point of p. Count-based windows can be deemed as a special case of time-based ones, where the window size W is measured in data objects, n is fixed for all slides, and a slide occurs after the arrival of a certain number of objects. The proposed methods are applicable to both types of windows. Some important definitions follow:

Definition 1. Object neighbors: *Let $R \geq 0$ be a user-specified threshold. For two data objects p_i and p_j, if the distance between them is no larger than R, p_i and p_j are said to be neighbors. The function $nn(p_i, R)$ denotes the number of neighbors that a data object p_i has, given the parameter R.*

Definition 2. Distance-Based Outlier: *Given R and a parameter $k \geq 0$, a distance-based outlier is an object p_i, where $nn(p_i, R) < k$.*

The set of distance-based outliers is denoted by $\mathcal{D}(R, k)$. If a point is not an outlier then it is an inlier. We represent the set of inliers by $\mathcal{I}(R, k)$. These two sets do not overlap and cover the complete object set, i.e., $\mathcal{D}(R, k) \cup \mathcal{I}(R, k) = \mathcal{P}$ and $\mathcal{D}(R, k) \cap \mathcal{I}(R, k) = \emptyset$.

Each pair of R and k parameters forms a query. Based on the above, the definition of the first problem we deal with, which refers to a single query, is as follows:

Problem 1. Single-query Distance-Based Outlier Detection: *Given the parameters R and k, and a fixed window size W output the distance-based outliers between all non-expired objects at each window slide.*

In this work, we also investigate a generalization of the same problem for multiple queries, that is multiple pairs of R and k parameters. More specifically, we additionally consider the following problem:

Problem 2. Multi-query Distance-Based Outlier Detection: *Given a set $\mathcal{Q}$ of queries, output the distance-based outliers between all non-expired objects for each query $q_i \in \mathcal{Q}$ at each window slide.*

A naive solution to the problem of continuous detection of distance-based outliers over windowed data streams would involve keeping for each object $p \in \mathcal{P}$ the complete set of its neighbors. Clearly, such an approach is characterized by quadratic space requirements ($O(n^2)$) in the worst case; as such, it is practically infeasible for large windows. As stated in the previous section, two more efficient approaches to this problem have been proposed. According to [26], for each object p, it is sufficient to keep at most k preceding neighbors and just the number of its succeeding neighbors n_p^+ to detect the distance-based outliers $\mathcal{D}(R, k)$ for specific R and k. Furthermore, for each new object p_{new}, a range query with radius R is executed to determine p_{new}'s neighbors. For each such neighbor p_i, $n_{p_i}^+$ is increased by one. Additionally, $P_{p_{new}}$ is updated with all the neighbors found and $n_{p_{new}}^+$ is set to zero. At any time instance, the approach adopted by [26] to decide if an object p is an outlier involves the computation of P_p. The cost to compute the size of P_p that corresponds to objects that have not expired is $O(logk)$, which means that the cost for all objects is $O(nlogk)$. The approach in [27] reduces this cost to $O(n)$, as it continuously keeps the number of neighbors of an object for all window slides until its expiration. Because of that, the method in [27] has worst case space requirements $O(nW)$, as it maintains up to W counters for each object (for *Slide* equal to 1 time unit). In the worst case, the space needed can become equal to $O(n^2)$. Moreover, each of these counters may be updated multiple times before becoming obsolete. However, [27] can answer queries with multiple values of k.

In summary, the approach in [26] has acceptable memory requirements ($O(kn)$), negligible time requirements to update the information for each existing object due to the arrival of new objects and the expiration of old objects ($O(1)$ for each new object), and significant time requirements to produce outliers ($O(nlogk)$). On the other hand, the approach in [27] has high memory

requirements ($O(nW)$), high time requirements to update existing information due to changes in the window population ($O(nW)$ for each new object), and low time requirements to produce the actual outliers ($O(n)$). In addition, both approaches require a range query with regard to all current objects $\mathcal{P}$ for each new object's arrival. In this work, we aim to develop algorithms that have both low space and time requirements, and also do not rely on the execution of expensive range queries that consider the entire set $\mathcal{P}$. Our solutions have $O(n)$ space requirements and we prove that we cannot solve the problems with less space; however they are faster than the exact algorithms in both [26] and [27].

4. A Theoretical Space Lower Bound

We prove a space lower bound for discovering outliers in a stream of data. To do that, we use results from one round communication complexity [41] by reducing the problem of Set-Disjointness [42] to our problem. We denote the one-round randomized communication complexity of a function $f : X \times Y \to Z$ with error δ by $R^1_\delta(f)$. Let $[n]$ represent the set of all natural numbers from 1 to n. Assume a partition of $[n]$ in three sets $\{T_1, T_2, \{i\}\}$. That is, $T_1 \cap T_2 = \emptyset$, $i \notin T_1, T_2$ and $T_1 \cup T_2 \cup \{i\} = [n]$. In the problem of Set-Disjointness, Alice gets a random subset $X \subseteq T_1 \cup \{i\}$ and Bob gets a random subset $Y \subseteq T_2 \cup \{i\}$. It is suitable to represent sets X and Y by the bit vectors x and y respectively of size n, so that the i-th bit x_i of x is 1 if and only if $i \in X$. The same holds for Y as well. The Set-Disjointness problem is defined as a boolean function f as follows:

$$f(x, y) = 1, \text{if } x \cdot y \neq 0$$

$$f(x, y) = 0, \text{if } x \cdot y = 0$$

where $\cdot$ is the inner product of vectors in $\{0, 1\}^n$. The following lemma comes from [42].

Lemma 1. *The one-way randomized communication complexity for the problem of Set-Disjointness is* $R^1_\delta(n) = \Omega(n)$.

We prove a space lower bound on the following problem, which we call the *counting distance-based outlier problem.*

Definition 3. *Calculate the number of distance-based outliers for some distance R and for $k = 2$.*

By reducing the problem of Set-Disjointness to the counting problem for distance-based outliers we can prove that this problem requires linear space even if we employ randomization and approximation. Apparently, the problem of finding the number of outliers is at most as difficult as the problem of really discovering them since in this case we can just count them. Thus, any lower bound for the problem of Definition 3 transfers to the more general problem.

Let n points in the cartesian space of dimension 2 that are represented by vectors $p_i \in \mathbb{R}^2, 1 \leq i \leq n$. These points are located in the cartesian space on the line $y = 0$ so that the Euclidean distance $d(p_i, p_{i+1})$ for all $1 \leq i < n$ is $3R$. We represent this set of points by $\mathcal{P}$. Alice and Bob know these points. Alice adds a point $a_i \in \mathbb{R}^2$ to the disk defined by point p_i with radius R if $x_i = 0$. Let this set of points added by Alice be $\mathcal{A}$. Similarly, Bob adds a point $b_i \in \mathbb{R}^2$ to the disk defined by point p_i with radius R if $y_i = 0$. Let this set of points added by Bob be $\mathcal{B}$ and finally assume that $\mathcal{Q} = \mathcal{P} \cup \mathcal{A} \cup \mathcal{B}$.

The following observation is crucial for the proof.

Observation 1. *There is an outlier in $\mathcal{Q}$ for some radius R and $k = 2$ if and only if $f(x, y) = 1$.*

Proof 1. $\Rightarrow$ *Let the outlier in $\mathcal{Q}$ be p_i. Then, p_i can be an outlier only if $x_i = y_i = 1$. This means that $f(x, y) = 1$.*
$\Leftarrow$ *Assume that $f(x, y) = 1$ and let i be the i-th bit such that $x_i = y_i = 1$. Then, no point will be added by either Alice or Bob and thus in a radius of R there is not going to be another point. Thus, p_i is an outlier.*

Assume that the real number of outliers in set $\mathcal{Q}$ is z. We say that an algorithm (ϵ, δ)-approximates the distance-based outlier counting problem if this algorithm returns an estimate $\tilde{z}$ of the real number of outliers z such that

$(1-\epsilon)z \leq \tilde{z} \leq (1+\epsilon)z$ with probability of success equal to $1-\delta$. The following lemma proves the lower bound.

Lemma 2. *If there exists a streaming algorithm that uses space s and that (ϵ, δ)-approximates the distance-based outlier counting problem for $0 < \epsilon, \delta < 1$, then $\mathcal{R}^1_\delta(f) \leq s$.*

Proof 2. *Assume that $\mathcal{M}$ is an algorithm that uses s space and (ϵ, δ)-approximates the distance-based outlier counting problem. Alice simulates $\mathcal{M}$ on the input provided by the points $\mathcal{P} \cup \mathcal{A}$. As soon as the simulation finishes she sends the s bits of the work space of $\mathcal{M}$ to Bob and he continues with the execution by providing the points in $\mathcal{B}$. If $\mathcal{M}$'s estimation is > 0 then Bob outputs 1, otherwise if the output is 0 then Bob outputs 0 as well.*

The crucial note is that if $\mathcal{M}$ outputs 0 then this can by definition be an ϵ-approximation only of 0 outliers, while if the number of outliers is ≥ 1 then 0 cannot be a valid ϵ-approximation, for any bounded ϵ. Thus, Bob outputs 1 iff there is at least one outlier. In fact, by definition there are either 0 outliers or the number of outliers is 1. Observation 1 concludes the proof.

Thus, by Lemma 1 it follows that $s = \Omega(n)$. This means that if we wish to discover in one pass the outliers in a stream of size n, then effectively the used memory must be asymptotically as large as the size of the stream. The same can be also said in the case one employs a sliding window of size W. Lemma 2 implies that the memory size must be $\Omega(W)$. Note that the bound is based on sparse point sets. In case an assumption is made about the input distribution then the bound does not hold.

5. Outlier Detection Algorithms

In this section, we provide algorithms for the continuous detection of distance-based outliers. We start by describing our framework for detection of outliers. The event-based method schedules efficiently potential changes in the set of outliers. Based on this framework, we develop four algorithms for distance-based outlier detection.

The first algorithm, a simple approach, which comes in two flavors, maintains outliers when the radius R and the number of neighbors k is constant while the second and the third algorithms build on the first by allowing these parameters to vary dynamically. The fourth algorithm, builds on the previous algorithms and reduces considerably the number of range queries over a sequence of departures and arrivals in the data stream.

5.1. The Event-Based Approach

We are interested in tracking the outliers in a set of objects of a stream defined by a sliding window. In particular, a set of outliers is maintained subject to arrivals of new objects from the stream and departures of existing objects due to the restricted window size (either restricted with respect to time or with respect to number of objects). The arrival and departure of objects has the effect of a continuously evolving set of outliers. At only certain discrete moments, however, this set may change and an inlier becomes an outlier or vice-versa. Between these discrete moments, the set of outliers remains as is.

The effect of arrivals of objects is to turn existing outliers into inliers. On the other hand, the potential effect of departures is to turn inliers into outliers. However, the exact time of the departure of each object is prespecified (due to the sliding window) and thus we can plan in the future the exact moments in which one needs to check whether an inlier has turned into outlier. The exact time of arrivals is considered unknown and unpredictable.

Henceforth, an *event* is the process of checking whether an inlier becomes an outlier due to departure of objects from the window. The expiration time of the objects is known whether we talk about time-based windows (in this case a new object p has expiration time $now + \lceil \frac{W}{Slide} \rceil$) or for count-based windows (in this case p expires after a predefined number of new objects have arrived). Thus, the time stamp of an event depends on the expiration time of objects. This forces a total order on the events which can be organized in an *event queue*. An event queue is a data structure that supports efficiently the following operations:

- $findmin$: returns the event with the most recent time stamp (the most

recent event).

- *extractmin*: invokes a call to $findmin$ and deletes this event from the event queue.
- $increasetime(p, t)$: increases the time stamp of the event associated to object p by t. It is assumed that we are provided with a pointer to p and there is no need to search for it.
- $insert(p, t)$: inserts an event for object p into the queue with time stamp t.

These operations can be supported efficiently by a min-ordered *priority queue*. Employing a Fibonacci heap allows us to support these operations in $O(1)$ worst-case time as well as in $O(\log n)$, $O(1)$ and $O(1)$ amortized time respectively [43] (one can also get similar worst-case bounds [44]). Note that due to the min-order of the heap, these structures support the operation of *decreasetime* which, however, can be trivially changed to support the operation of *increasetime*.

The event-based method for outliers employs an event queue to efficiently schedule the necessary checks that have to be made when objects depart. Thus, the event queue accommodates inliers only, since these can be affected by the departure of an object. Arrivals of new objects result in potential updates of the keys of some objects in the event queue. Additionally, existing outliers are checked as to whether they have become inliers and thus they should be inserted in the event queue.

5.2. The Basic Algorithm

In a similar manner to [26], it is sufficient to maintain at most k preceding neighbors and the number of succeeding neighbors for each object to detect the distance-based outliers $\mathcal{D}(R, k)$ for specific R and k. The *preceding neighbors* P_p of an object p are all objects within distance $\leq R$ from p while their arrival time is $< p.arr$. Similarly, the *succeeding neighbors* S_p are those with arrival

time $> p.arr$. For the succeeding neighbors of p only their number n_p^+ needs to be stored. Note that, if object p has $\geq k$ succeeding neighbors then p will never become an outlier, and it is called a *safe inlier*. A safe inlier is not stored in the event queue. Assuming that p is an inlier but not a safe one, meaning that $n_p^+ < k$, then we need to store the $k - n_p^+$ most recent objects in set P_p. This is because, only these objects can affect the status of the object p as in total there are k neighbors (see Figure 1 for an example). All objects are stored in a structure that supports range queries efficiently (e.g., an M-tree [45]). In the following, we describe how the event based-scheme is applied.

Let p be an object and let $p.minexp = min\{p_i.exp | p_i \in P_p\}$ be the minimum expiration time of P_p. Assume that object p is an inlier (not a safe one) at the present time instance (now). The event corresponding to p gets a time stamp $p.ev$ equal to $p.minexp$ and thus p will be checked again as an outlier candidate in time $p.ev$.

There are two cases as to what triggers the processing of the event queue and the update of $\mathcal{D}(R, k)$. Based on how we process the arrival of new objects we get two variations of the proposed method, which handle the event queue in a different manner. In the first variation, termed LUE (Lazy Update of Events), when a new object p' arrives, then a range query is performed, and for all returned objects $p_i \in \mathcal{D}(R, k)$, $n_{p_i}^+$ is increased by one. If some object p_i gets k neighbors then it is inserted in the event queue setting the value of $p.ev$ accordingly. Additionally, the set $P_{p'}$ is constructed with size at most k. All objects $p_i \in \mathcal{I}$ returned by the range query have their $n_{p_i}^+$ values increased by one. Finally, if $n_{p'}^- < k$ then p' is an outlier and it is added to $\mathcal{D}(R, k)$. Otherwise, p' is added to the event queue. When an object departs, then an event may be triggered by invoking $extractmin$ which returns object x from the event queue such that $x.ev = now$. If $n_x^- + n_x^+ < k$ then object x becomes an outlier and is added to $\mathcal{D}(R, k)$ otherwise, $x.ev$ and P_x are updated and it is reinserted into the event queue. The pseudocode of these operations is given in Algorithm 1, Algorithm 2 and Procedure 1 respectively.

In the second variation, termed DUE (Direct Update of Events), the ar-

Algorithm 1: ARRIVAL (p, now)
p: the new object, now: the current time instance

```
1.  A ← result of range query w.r.t p.
2.  for each q ∈ A do
3.      n_q^+ ← n_q^+ + 1;
4.      if (q ∈ D(R,k) and (n_q^- + n_q^+ == k)) then
5.          remove q from D(R,k);
6.          if (n_q^- ≠ 0) then
7.              ev ← min{p_i.exp | p_i ∈ P_q};
8.              insert(q, ev);
12. P_p ← k neighbors with the highest expiration times;
13. if (nn_p < k) then
14.     add p to D(R,k);
15. else
16.     ev ← min{p_i.exp | p_i ∈ P_p};
17.     insert(p, ev);
19. add p to the data structure supporting range queries;
```

rival of the new object p' forces the recomputation of event times of objects inside the event queue. In particular, all computations are the same with the exception that all objects $p_i \in \mathcal{I}$ returned by the range query have their events time updated. In addition, for each such object p_i its set P_{p_i} is updated and finally checked whether it has become a safe inlier. This means, that for each such object an *increasetime* operation is performed which is not as expensive as *extractmin*. When an event is processed concerning object x due to the departure of another object, then this event will surely cause x to become an outlier. In this way, we managed to reduce the number of calls to *extractmin* by making calls to *increasetime*. The pseudocode of Procedure 1 changes slightly as follows. Lines 4 and 6-10 are removed from Procedure 1 (PROCESSEVENT) since each event corresponds to an outlier. Additionally, just below Line 11 in algorithm Arrival we should add some lines that recompute the new event time ev for q (if q is in the event queue) and call procedure $increasetime(q, ev - now)$.

In Figure 1 we depict an example of LUE in the two dimensional space for $k = 4$ and for some fixed R. Let the subscripts denote the order of arrival

Algorithm 2: DEPARTURE (p, now)
p: the departing object, now: the current time instance

1. remove p from the data structure supporting range queries;
2. call PROCESSEVENT(p,now);

Procedure 1: PROCESSEVENT (p, now)
p: the departing object, now: the current time instance

1. $x \leftarrow findmin()$;
2. **while** $(x.ev == now)$ **do**
3. $x \leftarrow extractmin()$;
4. remove expired objects from P_x;
5. **if** $(n_x^- + n_x^+ < k)$ **then**
6. add x to $\mathcal{D}(R, k)$;
7. **else**
8. $ev \leftarrow min\{p_i.exp | p_i \in P_x\}$;
9. $insert(x, ev)$;
11. $x \leftarrow findmin()$;

of these objects. We focus on objects p_8 and p_{14} since all other nodes can be handled similarly. For object p_8 all objects p_i with $i < 8$ are preceding and all objects p_j with $j > 8$ are succeeding objects. Objects in the current window are denoted with black dots. In this example, $n_{p_8}^+ = 2$, $P_{p_8} = \{p_1\}$ and thus p_8 is an outlier. Similarly, $n_{p_{14}}^+ = 0$, $P_{p_{14}} = \{p_1, p_7, p_{10}, p_{12}\}$ and thus p_{14} is an inlier. Assume that object p_{22} arrives and after the range query we get $\mathcal{A} = \{p_8\}$. Then, $n_{p_8}^+$ is increased by one and thus p_8 gets four neighbors and becomes an inlier. Thus, $n_{p_8}^+ = 3$, $P_{p_8} = \{p_1\}$ and p_8 is inserted in the event queue with $p_8.ev = p_1.exp$. For p_{22} we have that $n_{p_{22}}^+ = 0$, $P_{p_{22}} = \{p_8\}$ and as a result it is an outlier. Finally, the event queue must be checked to find out whether some object has become an outlier again. Assume that the first object is expired, thus a Departure operation is invoked for object p_1. The event queue is checked and in this simple setting the object with the minimum event time is p_{14}. Thus, after the changes we get that $n_{p_{14}}^+ = 0$, $P_{p_{14}} = \{p_7, p_{10}, p_{12}\}$ and p_{14} becomes

an outlier. The process continues in the same way until an event is found for which the event time is $> now$. For DUE, a similar procedure is followed, with the exception that we process the event queue differently.

The complexity analysis of the two variants is presented in [28], where it is shown that LUE is preferred over DUE when the distribution of objects is very dense, while, if the object distribution is not very dense, then the second variation is preferred because it handles object departure more efficiently.

5.3. Multiple Outlier Detection

In a multi-user scenario, multiple queries with varying values of R and k may be posed. Each pair of R and k determines a query q of distance-based outlier detection. $\mathcal{D}(q.R, q.k)$ denotes the outliers of query q from the set of all queries $\mathcal{Q}$. In this section, we study the continuous evaluation of multiple queries. For simplicity, we discuss separately the case in which k varies and R remains constant and vice-versa. At the end, we combine trivially both methods into one so that both parameters can vary.

First, we examine the case where R is fixed and k varies. This means that all the valid queries $\mathcal{Q}$ have the same R and different values for the parameter k. The neighbors of an object are the same for all queries since R is fixed. Therefore, n_p^+ for an object p is the same for all queries. Moreover, for a query q, the value of n_p^- of an object p is at most $q.k - n_p^+$. Thus, the only possible

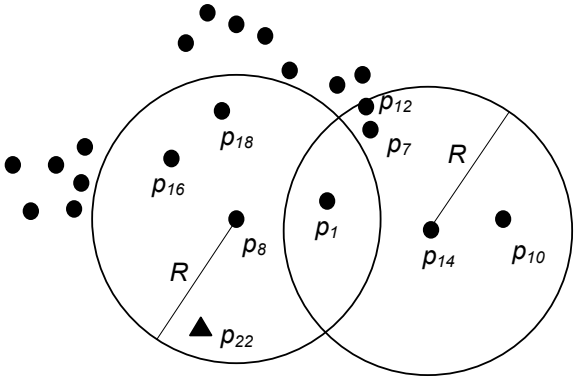


Figure 1: Example of LUE variant.

difference between queries is the size of P_p with respect to object p. Notice that, for two queries q_i and q_j, it holds that if $q_i.k < q_j.k$ then $\mathcal{D}(R, q_i.k) \subseteq \mathcal{D}(R, q_j.k)$. Therefore, if $k_{max} = max\{q_i.k\}$ $(0 \leq i \leq |\mathcal{Q}|)$, by keeping $k_{max} - n_p^+$ preceding neighbors for an object p, we can answer any query with $k \leq k_{max}$.

The algorithms are similar to the ones discussed in the previous section (both variations). Here we only report the changes. We continuously evaluate the query with the maximum value of parameter k, as described in the previous section. When an object departs, if the examination of an object p, at $p.ev$ time instance, reports p as outlier we check the other queries in $\mathcal{Q}$ whether p is also outlier in them. In particular, for each query q, if $n_p^- + n_p^+ < q.k$, then p is outlier in q. Queries are examined with decreasing order of k, and this procedure is terminated as soon as we reach a query for which p is inlier. Moreover, when a new object arrives, if object $p \in \mathcal{D}(R, q.k_{max})$ and its counter n_p^+ is increased, we check all the queries for a possible move of p from outlier set to inlier set. Notice that p is not necessarily outlier in all queries. For each query q, if $p \in \mathcal{D}(R, q.k)$ and $n_p^- + n_p^+ \geq q.k$ then p should be removed from $\mathcal{D}(R, q.k)$. The queries are examined again in decreasing order of k and the procedure is stopped when we reach a query in which p is not outlier. We call this algorithm COD (Continuous Outlier Detection).

We proceed now with the examination of the case of fixed k and varying R. In this case, two sets for each object p are maintained, the sets P_p and S_p (recall that we only stored the size of S_p) along with their distances from p, by taking into account the maximum distance $R_{max} = max\{q_i.R\}$ $(0 \leq i \leq |\mathcal{Q}|)$. When R varies it is necessary to maintain S_p since the neighbors of an object depend on the radius of the query. This may lead to high memory requirements, since in the worst case the number of neighbors can reach the number of active objects n. In the sequel, we study a more efficient scheme in terms of memory requirements. Therefore the size of S_p is limited to k objects.

The decision for the set of preceding neighbors is more difficult because both the nearest and most recent objects are preferable. If we keep the most recent objects, then it is possible to erroneously omit a neighbor, which affects the

query answer, with $q.R < R_{max}$ and if we keep the nearest objects it is possible to mistakenly report object p as an outlier when one of its nearest objects expires.

The key idea is the observation that all the preceding neighbors of p, which may have an impact on whether p is outlier or not, belong to the answer of the $k-1$-skyband query in the expiration time - distance space. A k'-skyband query reports all the objects that are dominated by at most k' other objects [46]. Therefore 0-skyband equals to the skyline query. In our case, the maximization of the expiration time and the minimization of the distance determine the domination relationship between objects, i.e., an object dominates another object if it has greater expiration time and smaller distance from p. The rationale of this observation is that at each time instance, the k nearest objects to p belong to the $(k-1)$-skyband of the preceding neighbors.

Therefore, when a new object arrives, the preceding neighbors are detected by taking into account the maximum distance R_{max}. Then, these objects are transformed to the expiration time - distance space. The objects belonging to the (k-1)-skyband are stored in P_p. Each entry of P_p consists of both the distance and the expiration time of the object.

Notice that the evaluation of the skyband query is required only once, when the object p arrives and P_p is initialized. [1] Then, it is sufficient to discard the expired objects. Moreover, if there are $n_p^{'+}$ ($< n_p^+$) succeeding neighbors of p with distance less than or equal to R_{min} then we can reduce the preceding neighbors that we keep in those which belong to the answer of the $(k-1-n_p^{'+})$-skyband query. This is because of the fact that if we have $n_{p_i}^{'+}$ succeeding neighbors for all the queries (since the distance from p is less than or equal to R_{min}) then the maximum number of preceding neighbors that could be used is $k - n_p^{'+}$. During the event processing, we can update the P_p set without

[1] For reasonable values of R_{max}, we expect that the number of neighbors with distance less than or equal to R_{max} will be much less than the number of active objects. For example, for 200K active objects from Zillow, by using R_{max} such that 1% of objects are outliers, on average only 561 objects belong to P_p (0.281% of $\mathcal{P}$).

evaluating the $(k-1-n_p^{'+})$-skyband from scratch, since the $(k-1-n_p^{'+})$-skyband is subset of the $(k-1)$-skyband. If $n_p^{'+} \geq k$ then no preceding neighbors are stored ($n_p^- = 0$). The following theorem guarantees the correctness of the algorithm (its proof is given in [28]).

Theorem 1. *Given the $n_p^{'+}$ succeeding neighbors with distance less than or equal to R_{min} for each object p, the distance-based outliers $\mathcal{D}(R,k)$ can be detected by keeping the $(k-1-n_p^{'+})$-skyband of the preceding neighbors of each object, if $(n_p^{'+} < k)$ or no preceding neighbors, if $(n_p^{'+} \geq k)$.*

To support the evaluation of multiple queries with different R we continuously evaluate the query with the minimum distance R_{min} because $\forall R > R_{min}, \mathcal{D}(R,k) \subseteq \mathcal{D}(R_{min},k)$. The event-based technique is used. Similarly to the case of varying k, if the examination of an object p, causes the move of p from the inliers to outliers then we should check p for the remaining queries with ascending order of R. The procedure is stopped when p is not moved to the outliers of a query. Moreover, when the set of succeeding neighbors of an outlier p increases due to the arrival of a new object, then we should check if p should be moved from outliers to inliers. Again all queries are examined with ascending order of R and the termination condition is similar.

In cases where both R and k are varying, we follow the latter methodology and we assume k equals to k_{max}. We evaluate the query with $q.R = R_{min}$ and $q.k = k_{max}$, because its outliers is a superset of the outliers of any other query. Finally, we filter the results with respect to each query q to provide the exact outliers. This algorithm is denoted as ACOD (Advanced Continuous Outlier Detection). In general, instead of searching the queries in decreasing order of k (resp. increasing order of R) we can perform a binary search to identify the value of k (resp. of R) for which an object becomes inlier. It holds that if an object belongs to $\mathcal{D}(R,k)$, it also belongs to $\mathcal{D}(R',k')$ for any $R' \leq R$, $k' \geq k$. Similarly, if object belongs to $\mathcal{I}(R,k)$, it also belongs to $\mathcal{I}(R',k')$ for any $R' \geq R$, $k' \leq k$.

5.4. Mitigating the Impact of Range Queries

The previously proposed methods provide an efficient way to perform potentially multi-parameter distance-based outlier detection. Nevertheless, they still suffer from a significant limitation, which characterizes all proposals to date for outlier detection in streams, namely the need to evaluate range queries for each new object with respect to all other active objects [27, 26]. In this section, we propose a methodology to mitigate this. Our methodology is based on the concept of evolving micro-clusters that correspond to regions containing inliers exclusively. The resulting algorithm is denoted as MCOD (Micro-cluster-based Continuous Outlier Detection). The additional symbols used are presented in Table 2.

Let us assume that, initially, the R and k parameters for outlier detection are fixed. We set the radius of MC_i, which is the maximum distance of any object belonging to MC_i from mcc_i, to $R/2$, and the minimum size of a micro-cluster to $k+1$. An object can belong to at most a single micro-cluster. As such, there are at most $\lfloor n/(k+1) \rfloor$ micro-clusters at any window. In general, an object may have neighbors that belong to other micro-clusters. However, the centers of such micro-clusters are within a range of $2R$ from each other.

Note that micro-clusters have been employed in several works to assist clustering in streamed data [47, 48]. Such works tend to build upon the cluster feature vector introduced in [5], to attain a more compact representation of the objects with a view to improving clustering efficiency without sacrificing clus-

Table 2: Additional symbols used in MCOD

Symbol	**Interpretation**
MC_i	the i-th micro-cluster
mcc_i	the center of the i-th micro-cluster
mcn_i	the size of MC_i, i.e., the number of objects it contains
$p.mc$	the identifier of the micro-cluster of object p
$p.Rmc$	the list of micro-clusters associated with object p
$\mathcal{I}^{mc}$	the set of objects that belong to a micro-cluster
$\mathcal{PD}$	the set of objects that do not belong to any micro-cluster

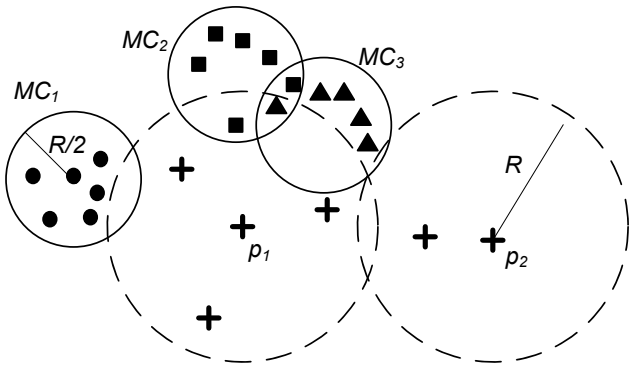


Figure 2: Some micro-clusters for k=4.

ter quality. The actual clustering is performed by a subsequent offline stage. However, in our case, micro-clustering serves a different purpose, i.e., outlier detection, and micro-clusters are fully tailored to online processing.

In the example of Figure 2, there are three micro-clusters, and for the objects of each one of them, a different symbol has been used. When the micro-clusters are thought of as spheres with radius $R/2$, they can be either overlapping (e.g., $MC_2,\ MC_3$) or not-overlapping (e.g., MC_1). Even in the former case, an object always belongs to a single micro-cluster, as explained later. Moreover, the center of the micro-cluster may correspond to an existing object (e.g., MC_1) or may not (e.g., $MC_2,\ MC_3$); the center does not change to eliminate the need to reconsider micro-cluster population at runtime. We regard all objects in $\mathcal{PD}$ as potential outliers (e.g., $p_1,\ p_2$); such objects are depicted with the + symbol. However, an object that does not belong to any micro-cluster may be an inlier (e.g., p_1).

Note that all the following expressions hold: $\mathcal{I}^{mc} \subseteq \mathcal{I} \subseteq \mathcal{P}$, $\mathcal{D} \subseteq \mathcal{PD} \subseteq \mathcal{P}$, $\mathcal{I}^{mc} \cup \mathcal{PD} = \mathcal{P}$, and $\mathcal{I}^{mc} \cap \mathcal{PD} = \emptyset$. The following two lemmas hold; the corresponding proofs can be found in [28].

Lemma 3. *An object that belongs to a micro-cluster (i.e., $p \in \mathcal{I}^{mc}$) is definitely not an outlier.*

Algorithm 3: UPDATE-MCOD (p, p', now)
p: the arriving object, p': the expired object
now: the current time instance

1. make a $(\frac{3}{2}R)$-range query to the centers of clusters w.r.t. p;
   Let $\mathcal{C}$ the set of clusters returned and MC_c the closest cluster;
2. **if** $(distance(p, mcc_c) \leq \frac{R}{2})$ **then**
3. $p.mc = MC_c$; $mcn_c = mcn_c + 1$;
4. $\mathcal{A} = \{q | q \in \mathcal{PD} \vee MC_c \in q.Rmc\}$;
5. **for each** $q \in \mathcal{A}$ **do**
6. **if** $(distance(q, p) \leq R)$ **then**
7. $n_q^+ = n_q^+ + 1$;
8. **if** $(q \in \mathcal{D}(R, k)$ **and** $(n_q^- + n_q^+ == k))$ **then**
9. remove q from $\mathcal{D}(R, k)$;
10. **else**
11. make an R-range query to objects $\in \mathcal{PD}$ w.r.t. p;
    Let $\mathcal{A}$ the set of objects returned;
12. **for each** $q \in \mathcal{A}$ **do**
13. **if** $(distance(q, p) \leq R)$ **then**
14. insert q to P_p;
15. $n_q^+ = n_q^+ + 1$;
16. **if** $(q \in \mathcal{D}(R, k)$ **and** $(n_q^- + n_q^+ == k))$ **then**
17. remove q from $\mathcal{D}(R, k)$;
18. **if** $(distance(q, p) \leq \frac{R}{2})$ **then** insert q to $\mathcal{NC}$;
19. **else** insert q to $\mathcal{NNC}$;
20. **if** $(|\mathcal{NC}| \geq \theta \cdot k)$ **then** $//\theta \geq 1$
21. create new cluster MC_n; $mcc_n = p$; $mcn_n = |\mathcal{NC}|$;
22. **for each** $q \in \mathcal{NC}$ **do**
23. $q.mc = MC_n$;
24. move q from $\mathcal{PD}$ to $\mathcal{I}^{mc}$;
25. **for each** $q \in \mathcal{NNC}$ **do** insert MC_n to $q.Rmc$
26. **else**
27. **for each** $q \in \mathcal{C}$ **do**
28. $ev = min\{p_i.exp | p_i \in P_p\}$;
29. $insert(p, ev)$;
30. **if** $(p' \in MC_o)$ **then**
31. $mcn_o = mcn_o - 1$;
32. **if** $(mcn_o < k)$ **then**
33. remove MC_o from clusters;
34. **for each** $q \in MC_o$ **do**
35. treat q as new object without
    updating its neighbors
36. remove p' from the data structure supporting range queries;
37. call PROCESSEVENT(now);

Lemma 4. *An object p belongs to the set of outliers $\mathcal{D}$ if and only if there are less than k neighbors of p in either the set of potential outliers $\mathcal{PD}$ or in $\mathcal{I}^{mc}$, such that the distance from the center of those micro-clusters is at most $\frac{3}{2}R$.*

The information kept for each object in the current window differs on the basis of the set it belongs to. More specifically, for objects $p \in \mathcal{I}^{mc}$, we only keep $p.mc$. For each object $p \in \mathcal{PD}$, we keep the expiration time of the k most recent preceding neighbors and the number of succeeding neighbors, as described in the previous sections. In addition, we keep a list containing the identifiers of the micro-clusters, whose centers are less than $\frac{3}{2}R$ far. The reason we keep this information derives from the lemma above. The assignment of objects to those micro-clusters may lead to a change in the status of the potential outliers; in other words, the micro-clusters of this type may affect the objects in $\mathcal{PD}$. The list of identifiers is stored in $p.Rmc$. Also, we employ a hash data structure so that we can find i) the objects in each micro-cluster, ii) the objects deemed as potential outliers, and iii) the objects in $\mathcal{PD}$ referring to a particular micro-cluster in $O(1)$ time.

The main rationale behind our approach is i) to drastically reduce the number of objects that are considered during the range queries when these are performed; and ii) the event queue not to include objects that belong to I^{mc}. The pseudocode when a single object arrives and a single object departs is presented in Algorithm 3. It can be generalized for the case where multiple objects arrive and depart in a straight-forward manner. The detailed steps of the algorithm after each window slide and the proof of correctness are discussed in [28].

The efficiency of this algorithm is expected to improve proportionally with the size of $\mathcal{I}^{mc}$. In other words, if the size of $\mathcal{PD}$ is small, and close to the size of the actual outliers, then the performance improvements are expected to be higher. This is the case when the (average) density of the objects is higher than the density threshold implied by the R and k parameters by several factors.

Finally, this methodology can easily support multiple values for k, if the minimum size of micro-cluster is set to $k_{max} + 1$. Also, multiple values for

both R and k can be supported, if we maintain additionaly information for the potential outliers and if we use an appropriate size for the micro-clusters. More specifically, similarly to ACOD, for each potential outlier p we have to store a set of preceding neighbors along with their distances from p and a set of succeeding neighbors. The events of the potential outliers are determined and examined similarly to the events of ACOD. Moreover, the minimum cardinality for the micro-clusters is set to $k_{max}+1$ and the maximum allowed distance from their center is set to $R_{min}/2$. This algorithm is denoted as AMCOD and it is included in the performance results given in the following section.

6. Performance Evaluation

We have conducted a series of experiments to evaluate the performance of the proposed algorithms. We compare algorithms COD and MCOD against the algorithm in [27], which is termed Abstract-C. These three methods handle queries with different values of k and fixed R. Note that, we do not include the simple algorithm of Section 5.2 which requires k and R to be fixed, since its functionality is covered by COD algorithm. We also have studied the advanced algorithms ACOD and AMCOD which is used for multiple queries with different values of both k and R. All methods have been implemented in C++ and the experiments have been conducted on a Pentium@3.0GHz WinXP machine with 1GB of RAM. In addition, a JAVA implementation integrated into the MOA[2] framework [49] can be found in [50].

We have used two real-life and two synthetic data sets. The real data sets are i) FC (Forest Cover), available at the UCI KDD Archive (url:kdd.ics.uci.edu), containing 581,012 records with quantitative attributes such as elevation, slope etc. and ii) ZIL (Zillow), extracted from www.zillow.com, containing 1,252,208 records with attributes such as price and number of bedrooms. The first synthetic data set (IND) contains 5M objects with independent attributes that follow a uniform distribution. We also generated a more complicated synthetic

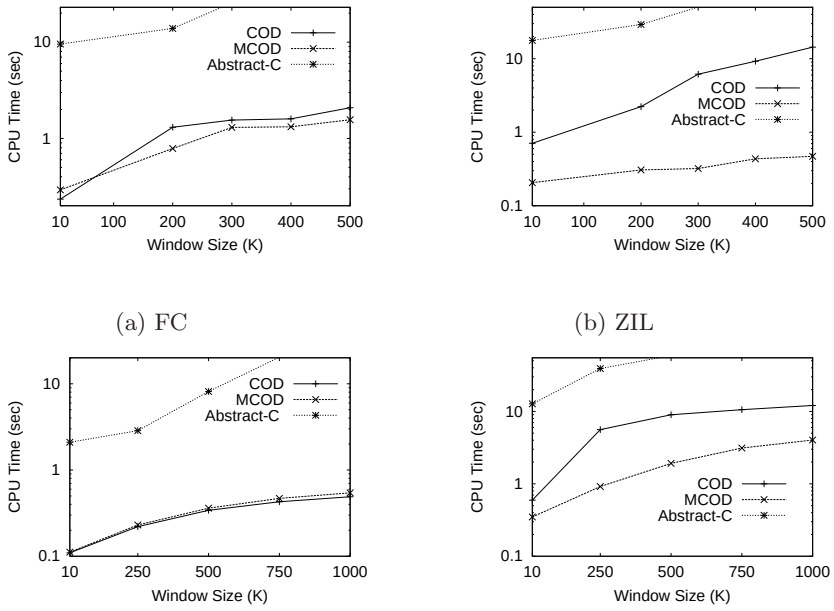
[2]http://moa.cms.waikato.ac.nz

data set (GAU). This set comprises of 5M objects; 60% of objects follow a uniform distribution and the remaining objects follow 4 different gaussian distributions of equi-sized population.

We study the performance of the proposed methods by varying the most important parameters such as the window size W, the distance R, the number of required neighbors k and the number of queries. We measure the CPU cost, memory requirements, the number of distance computations and other measurements. The default values for the parameters (unless explicitly specified otherwise) are: $W = n = 200K$, $|\mathcal{Q}| = 1$, i.e., there is a single query, $k = 10$ and the parameter R is set in a way that the number of outliers $|\mathcal{D}| = (0.01 \pm 0.001)n$. Since we want to investigate the most demanding form of continuous queries, we set $Slide = 1$, for the proposed methods. However, the memory requirements of Abstract-C are very high for $Slide = 1$. More specifically, Abstract-C stores $\frac{W \cdot (W-1)}{2}$ counters, which corresponds to 74GB for $W = 200K$ and 465GB for $W = 500$, assuming integers need 4 bytes. Because of that, we used $Slide = 1$ only for our proposed methods, while we choose $Slide = 0.001W$ for Abstract-C. All measurements correspond to 1000 slides, i.e., 1000 insertions/deletions.

6.1. Running Time

First, we study algorithms COD, MCOD and Abstract-C which can handle multiple queries with different values of k. The first experiment studies the performance of the methods for varying values of W in the range $[10K, 1000K]$. Figure 3 depicts the results. Despite the favorable configuration of parameter slide, Abstract-C performs significantly worse than our algorithms in terms of running time. The event-based technique used by COD benefit from the fact that not all objects need to be investigated at each slide. MCOD is even better because uses micro-clusters, therefore many objects are not investigated at each slide and also the number of range queries which is very consuming operation is reduced. In general, MCOD runs faster than COD because it reduces the number of distance computations by avoiding the application of a range query for each new object. However, the two methods have similar performance for



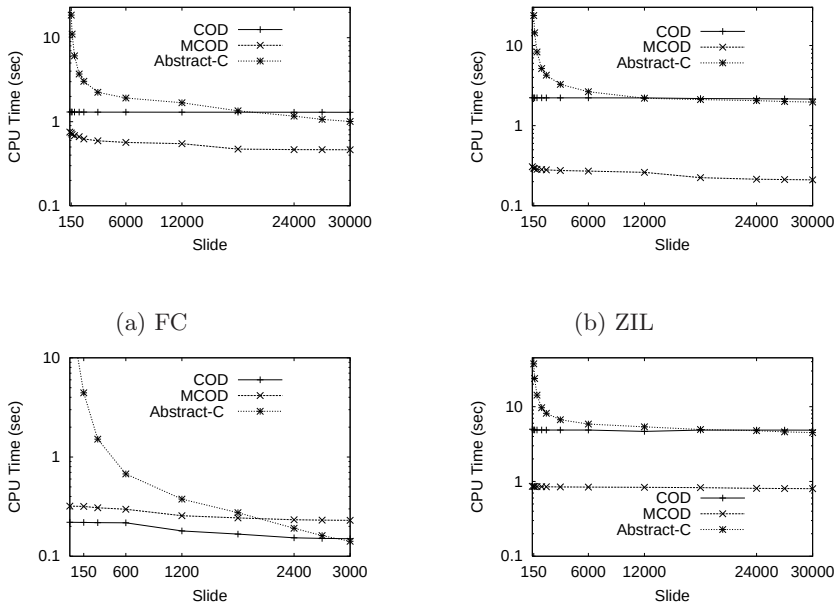
(a) FC (b) ZIL

(c) IND (d) GAU

Figure 3: Running time vs. active objects (COD, MCOD and Abstract-C).

the IND data set, where MCOD generates a negligible number of micro-clusters and therefore the method degenerates to COD. Also COD performs better than MCOD in small windows as shown for the FC dataset in Figure 3 because the overhead of micro-cluster maintenance exceeds the gain of distance computation reduction, since the absolute number of distance computations is low.

The performance of Abstract-C is affected drastically by parameter *Slide* both in response time and memory consumption. The next experiment studies the behavior of Abstract-C with respect to *Slide*. The results are presented in Figure 4. As expected, COD and ACOD are slightly affected while Abstract-C is improved as *Slide* increases. However, Abstract-C is better than COD only for very large slides and better than ACOD only for special cases of data distribution. As shown in Figure 4, Abstract-C outperforms COD in cases where *Slide* is more than 12% of the window size for FC, ZIL and GAU datasets and



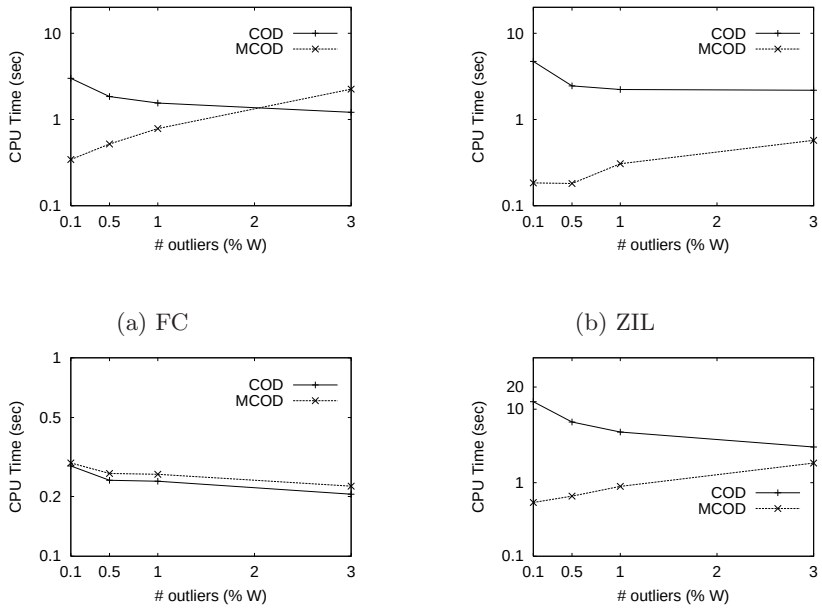
(a) FC (b) ZIL

(c) IND (d) GAU

Figure 4: Running time vs. slide (COD, MCOD and Abstract-C).

1.2%W for IND dataset. Moreover, ACOD is consistently better than Abstract-C except for the IND dataset, due to the poor ability of micro-cluster generation. From this experiment, it is evident that Abstract-C is rather inefficient for continuous outlier monitoring and can be used more efficiently in cases where outlier snapshots or approximate answers are adequate. Abstract-C is omitted from subsequent experiments. Note that we do not present experiments with [26], because its running time is worse than Abstract-C for continuous outlier detection, and its memory consumption is not lower than ours.

Next, we investigate the performance of the proposed methods with respect to the number of outliers. The results are given in Figure 5. The outliers' number varies from 0.1% to 3% of n, which is set to its default value 200,000. As expected, MCOD is better than COD in most of the cases, whereas, the performance of MCOD may degrade as the number of outliers increases.

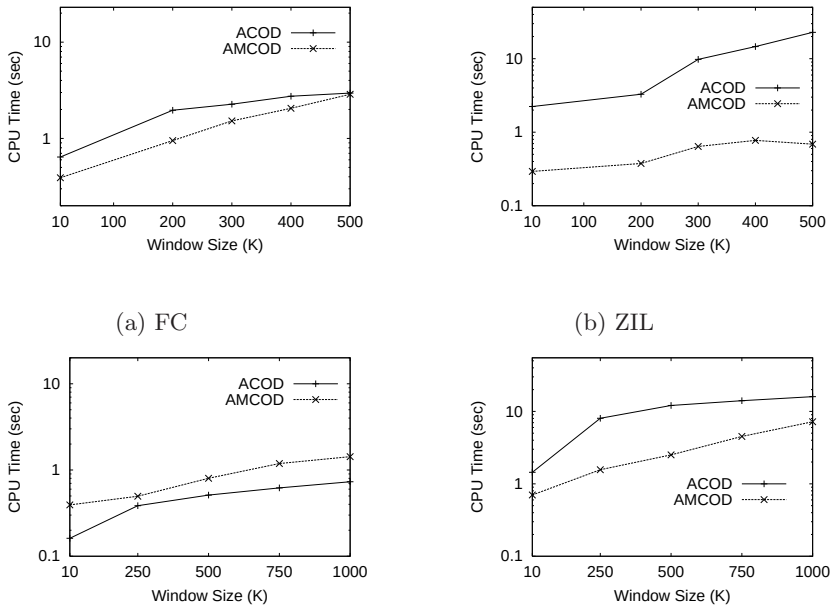


(a) FC (b) ZIL

(c) IND (d) GAU

Figure 5: Running time vs. number of outliers (COD, MCOD and Abstract-C).

In the next two experiments, we study the performance of ACOD and AMCOD. Figure 6 shows the running time of the methods for different values of W, whereas Figure 7 depicts the result for different number of outliers. Both experiments show the superiority of AMCOD against ACOD, in cases where the data distribution favors the micro-cluster generation. As expected, the performance of AMCOD is affected from its ability to generate micro-clusters, thus ACOD reduces the gap between them when either the data distribution does not allow efficient micro-cluster generation (i.e. uniform distribution) or the number of outliers increases. Further observations can be drawn from Figures 3 to 7. Methods COD and MCOD perform better than ACOD and AMCOD respectively since in both experiments there is only a single query, whereas, ACOD and AMCOD should be used only in cases of multiple queries with different R values.



(a) FC (b) ZIL

(c) IND (d) GAU

Figure 6: Running time vs. active objects (ACOD and AMCOD).

In the next experiment, we investigate multiple queries. Parameter R is fixed for all queries to examine the efficiency of the COD and MCOD, which can support different values only of k. ACOD and AMCOD are also reported for comparison reasons. More specifically, for IND, $R = 73.5$ while $k \in [5, 14]$, for FC, $R = 42$ and $k \in [5, 10]$, for ZIL, $R = 3600$ and $k \in [5, 10]$, whereas for GAU $R = 63$ and $k \in [5, 14]$. Figure 8 illustrates the CPU time of the algorithms. Notice that there may exist similar queries, due to the limited number of different values of k. However, to better examine scalability, the methods do not exploit the existence of similar queries. As mentioned before, ACOD and AMCOD are appropriate for varying values of R and therefore they present the worst performance. It is evident that the running time of all methods increases sublinearly with respect to the number of queries. Again, MCOD is better than COD except for IND data set, for which it generates only a few

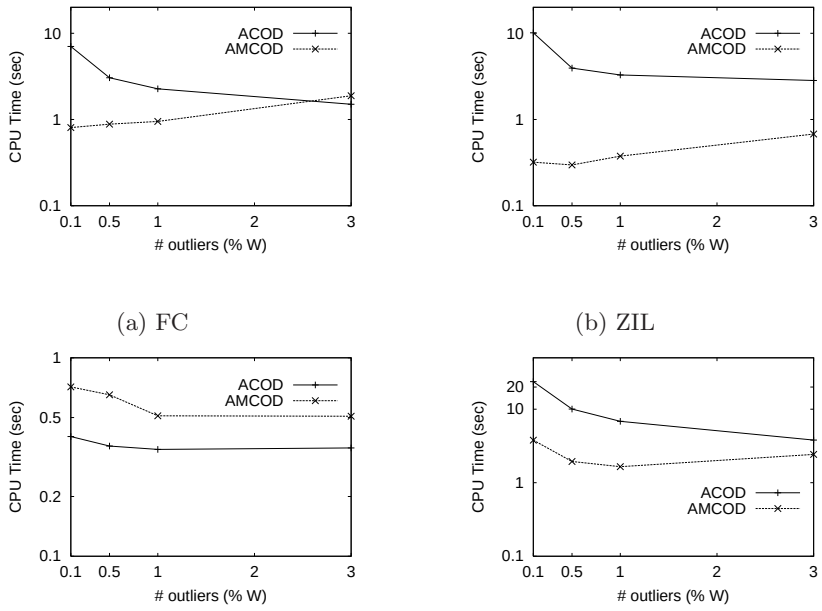
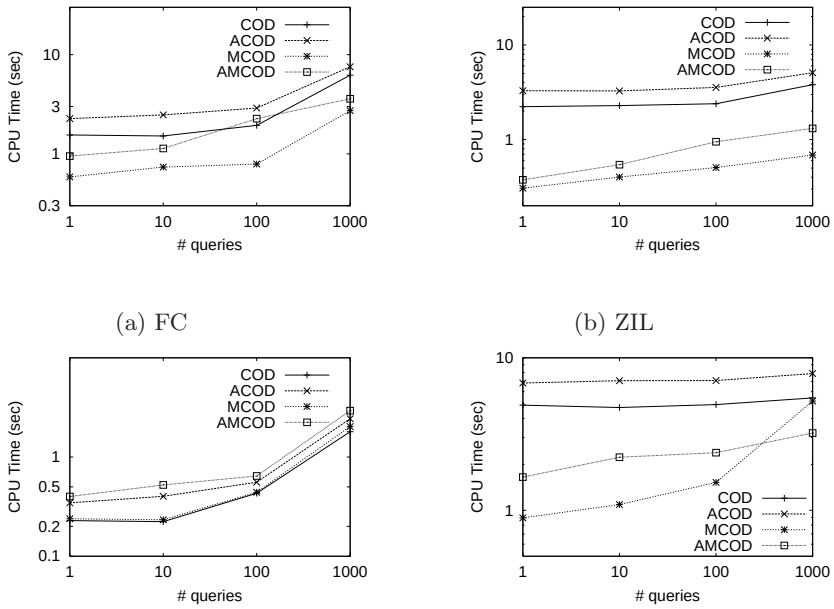


Figure 7: Running time vs. number of outliers (ACOD and AMCOD).

micro-clusters.

The next experiment studies the usability of ACOD and AMCOD by varying the number of queries while allowing different values for both R and k. The methods COD and MCOD are used for comparisons reasons. ACOD and AMCOD evaluate all the queries together whereas COD and MCOD evaluate each query separately and the sum of all the running times is presented. Figure 9 shows the results. ACOD performance improves as the number of queries increases. Although the evaluation of the query with k_{max} and R_{min} with the ACOD method is the most time consuming, ACOD has the best performance because of result reuse for the remaining queries. AMCOD is better that ACOD in most of the cases. However, as k_{max} increases and R_{min} decreases, the number of outliers increases for the basic query and therefore the generation of micro-clusters degrades. In these cases, AMCOD performs worse than ACOD.



(a) FC (b) ZIL

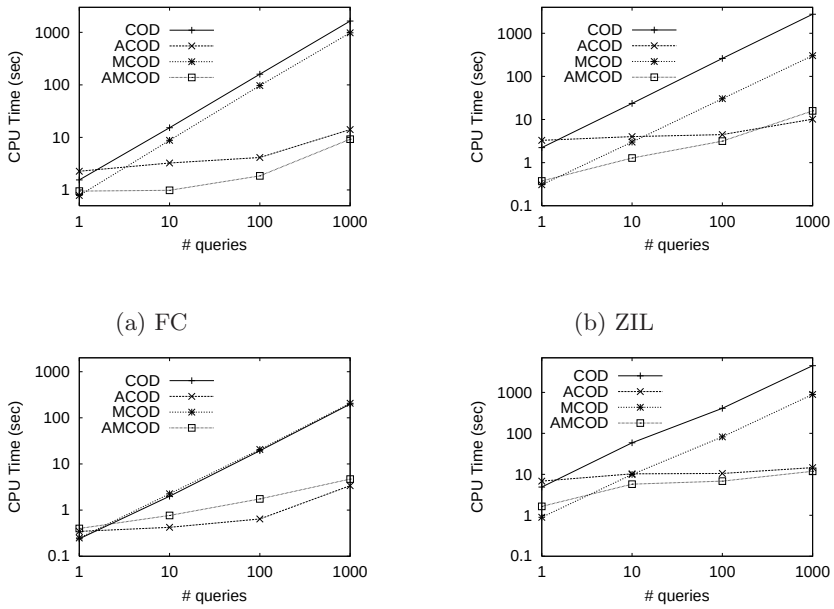
(c) IND (d) GAU

Figure 8: Running time vs. number of queries with different values of k.

6.2. Qualitative Analysis

First, we examine the behavior of the event-based technique. For each method, the following measurements are taken: a) the average number of events that exist in the system, b) the average number of events triggered by the arrival of new objects, and c) the average number of events processed after each arrival. Table 3 illustrates the results for the FC data set in the experiments of Figures 5 and 7.

Notice that the number of events triggered is more than the number of object processed. This is because the former includes also events related to expired objects and events corresponding to objects that have become safe inliers. These events are immediately discarded without any further processing and only the remaining events are processed. Event processing includes the update of the neighbors of the object, the check for possible inclusion of the object to the



(a) FC (b) ZIL

(c) IND (d) GAU

Figure 9: Running time vs. number of queries with different values of R and k.

outliers and the re-estimation of the event for the specific object. From the fourth and the fifth column of the table, it is evident that the majority of events are discarded immediately. In COD, the number of events is very close to the number of objects not in the outliers set, thus the events are reduced as the number of outlier increases, contrary to the behavior of ACOD, MCOD and AMCOD. For the other data sets, the total number of events is similar but less events are processed, due to the fact that the average number of an object's neighbors is higher and more objects become safe inliers.

The next table presents the affect of micro-cluster technique for the ZIL data set for the experiment of Figure 6. The first column for each method (CPU RQ) shows the CPU time consumed for the processing of the range queries. As mentioned before, the CPU time corresponds to 1000 slides. The second column (DC) shows the total number of distance computations including the

Table 3: Event Analysis (FC data set).

outliers (%W)	algorithm	#events (K)	#events triggered (avg)	#events processed (avg)
0.1	COD	198.5	1.3	0.19
	ACOD	144.4	9.6	0.55
	MCOD	16.1	1.1	0.19
	AMCOD	8.6	2.2	0.44
0.5	COD	196.2	1.9	0.67
	ACOD	145.4	10.3	1.49
	MCOD	46.6	1.9	0.67
	AMCOD	28.3	5.1	1.39
1	COD	194.5	2.4	1.09
	ACOD	146.4	10.6	2.04
	MCOD	65.9	2.5	1.08
	AMCOD	43.1	6.5	1.92
3	COD	189.5	3.3	1.61
	ACOD	149.3	10.2	2.49
	MCOD	113.3	3.5	1.61
	AMCOD	84.8	8.3	2.35

distance computations between objects, between clusters and between objects with clusters. It is evident that micro-cluster technique reduces drastically the number of range queries and therefore the number of distance computations which finally affects the CPU consume.

6.3. Memory Consumption

Table 5 presents the memory consumption of the two real data sets for the experiments of Figures 3 and 6. The consumed memory corresponds to the memory needed to store the information for each active object (i.e., preceding and succeeding neighbors), the heap size used for the events prioritization, the outliers of all the queries and the micro-cluster information for MCOD. As can be seen, the required amount of memory is only a small fraction of the total

Table 4: Micro-cluster affect.

	ZIL			
	ACOD		AMCOD	
W	CPU RQ (sec)	DC (K)	CPU RQ (sec)	DC (K)
10,000	1.04	4032.8	0.08	61.0
200,000	2.52	13663.9	0.17	269.2
300,000	6.98	35698.6	0.22	425.8
400,000	10.25	50593.1	0.27	548.5
500,000	16.06	81816.6	0.27	577.9

Table 5: Memory requirements (in MBytes).

	FC				ZIL			
W	COD	ACOD	MCOD	AMCOD	COD	ACOD	MCOD	AMCOD
10,000	0.46	2.95	0.27	0.27	0.48	4.29	0.11	0.10
200,000	9.58	100.45	4.94	5.49	9.60	111.85	2.74	3.11
300,000	14.11	133.27	11.04	13.12	14.47	194.28	4.01	4.61
400,000	18.76	178.30	15.40	18.78	19.32	280.37	5.43	6.32
500,000	23.52	232.72	20.23	25.15	24.23	377.51	6.56	7.66

memory available in modern machines, even for the ACOD method. However, AMCOD achieves better performance than ACOD and uses much less amount of memory, slightly more memory than that of MCOD.

Table 6 shows the average number of neighbors kept in lists S and P for the same experiments for FC data set. The sum of cardinality of S and P is presented. Notice that COD and MCOD have only preceding neighbors whereas ACOD and AMCOD have preceding and succeeding neighbors. ACOD keeps larger number of neighbors than COD and uses this information to reduce the number of events processes. MCOD and AMCOD stores less neighbors than COD and ACOD respectively due to the use of micro-clusters. Many objects are not associated with an event, therefore no neighbors are maintained for these objects. Note that memory consumption is affected by both the size of these lists and the size of the events, thus the overall gain of MCOD and AMCOD regarding the memory consumption is higher.

Table 6: Average number of neighbors maintained per object.

	FC			
W	COD	ACOD	MCOD	AMCOD
10,000	9.2	32.2	1.7	5.0
200,000	9.6	51.8	1.2	4.7
300,000	9.4	45.6	2.5	9.1
400,000	9.4	45.8	2.8	10.5
500,000	9.4	47.8	2.8	11.2

6.4. LUE vs. DUE

In all the previous experiments the first variation (LUE) of event handling is used. Table 7 compares experimentally both variants of event handling as described in Section 5.2. for COD. For each variant, we measure a) the average number of events that exist in the system, b) the average number of events triggered by each arrival, c) the average number of events inserted in the queue after each arrival and d) the average number of *increasetime* operations in each update.

Table 7: Event Handling Variations (W = 200K, outliers = 1%W).

	IND		FC		ZIL	
	LUE	DUE	LUE	DUE	LUE	DUE
#events (in K)	164.4	79.9	194.5	7.50	195.8	9.91
#events triggered	6.07	0.12	2.37	0.92	1.58	1.38
#events inserted	1.55	1.10	1.91	1.74	1.16	1.11
#*increasetime* ops	-	9.81	-	10.1	-	9.57

DUE has much better space usage because all objects that are safe inliers are removed straight away from the event queue. This is more tense when the distribution of objects is skewed as in the cases of FC and ZIL. The number of triggered events (and as a result the number of (re)inserted events) is lower in DUE than in LUE. This was expected, since DUE focuses on reducing these costly operations and replacing them with *increasetime* operations, which are theoretically cheaper. Note that due to the smaller size of the event queue in DUE, the operation of *extractmin* (event trigger) is cheaper and thus the savings are twofold. However, the implementation of the event queue in DUE is much more complicated and thus, these operations have larger absolute cost. Although further experimental analysis is needed to clarify in which setting each algorithm is better, DUE is more preferable in cases where the available memory is limited.

6.5. Summary of Results

We conducted extensive experiments in both synthetic and real datasets. The results show the efficiency and the effectiveness of the proposed methods.

The proposed algorithms are consistently more efficient than Abstract-C, the state-of the art online algorithm for continuous outlier detection. Abstract-C may perform better only for very large values of parameter *Slide*, thus is more appropriate for applications where snapshots of outliers or approximate answers without guarantees are required (i.e., we depart from the continuous outlier detection problem). Moreover, MCOD and AMCOD typically perfrom better than COD and ACOD, respectively. However, there are two cases where COD and ACOD are more suitable: a) when the data distribution does not favor the micro-cluster generation, as in the IND dataset and b) when the number of outliers is very high with respect to the active window size.

7. Conclusions

Anomaly detection is an important data mining task aiming at the selection of some interesting objects, called outliers, that show significantly different characteristics than the rest of the data set. In this work, we study the problem of continuous outlier detection over data streams, by using sliding windows. More specifically, four algorithms are designed, aiming at efficient outlier monitoring with reduced storage requirements. Our methods do not make any assumptions regarding the nature of the data, except from the fact that objects are assumed to live in a metric space. As it is shown in the performance evaluation results, based on real- life and synthetic data sets, the proposed techniques are by factors more efficient than previously proposed algorithms.

There are several directions for future research. It is interesting to design outlier detections algorithms over uncertain data streams, where each object has an assigned existential probability. A second direction is to use load shedding techniques in outlier mining towards performance improvement.

References

[1] R. Johnson, Applied Multivariate Statistical Analysis, Prentice Hall, 1992.

[2] E. Knorr, R. Ng, Algorithms for mining distance-based outliers in large data sets, in: VLDB, 1998.

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

[4] S. Muthukrishnan, Data streams: Algorithms and applications, Found. Trends Theor. Comput. Sci. 1 (2005).

[5] T. Zhang, R. Ramakrishnan, M. Livny, Birch: An efficient data clustering method for very large databases, in: SIGMOD, 1996, pp. 103–114.

[6] C. C. Aggarwal, Outlier Analysis, Springer, 2013.

[7] M. Gupta, J. Gao, C. C. Aggarwal, J. Han, Outlier detection for temporal data: A survey, IEEE Trans. Knowl. Data Eng. 26 (2014) 2250–2267.

[8] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection for discrete sequences: A survey, IEEE Trans. Knowl. Data Eng. 24 (2012) 823–839.

[9] A. Zimek, E. Schubert, H. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, Statistical Analysis and Data Mining 5 (2012) 363–387.

[10] L. Akoglu, H. Tong, D. Koutra, Graph based anomaly detection and description: a survey, Data Min. Knowl. Discov. 29 (2015) 626–688.

[11] V. Barnett, T. Lewis, Outliers in Statistical Data, Wiley and Sons, 1994.

[12] C. Aggarwal, P. Yu, Outlier detection for high dimensional data, in: SIGMOD, 2001, pp. 37–46.

[13] Y. Tao, X. Xiao, S. Zhou, Mining distance-based outliers from large databases in any metric space, in: SIGKDD, 2006, pp. 394–403.

[14] M. Breunig, H.-P. Kriegel, R. Ng, J. Sander, Lof: Identifying density-based local outliers, in: SIGMOD, 2000.

[15] X. H. Dang, I. Assent, R. T. Ng, A. Zimek, E. Schubert, Discriminative features for identifying and interpreting outliers, in: ICDE, 2014.

[16] E. Wu, S. Madden, Scorpion: Explaining away outliers in aggregate queries, in: VLDB, 2013.

[17] M. Henrion, D. J. Hand, A. Gandy, D. J. Mortlock, CASOS: a subspace method for anomaly detection in high dimensional astronomical databases, Statistical Analysis and Data Mining 6 (2013) 53–72.

[18] N. Pham, R. Pagh, A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data, in: KDD, 2012, pp. 877–885.

[19] B. Perozzi, L. Akoglu, P. I. Sanchez, E. Müller, Focused clustering and outlier detection in large attributed graphs, in: KDD, ????, pp. 1346–1355.

[20] M. Mongiovì, P. Bogdanov, R. Ranca, A. K. Singh, E. E. Papalexakis, C. Faloutsos, Sigspot: mining significant anomalous regions from time-evolving networks (abstract only), in: SIGMOD, 2012, p. 865.

[21] L. Akoglu, H. Tong, J. Vreeken, C. Faloutsos, Fast and reliable anomaly detection in categorical data, in: CIKM, 2012, pp. 415–424.

[22] E. Schubert, A. Zimek, H. Kriegel, Local outlier detection reconsidered: a generalized view on locality with applications to spatial, video, and network outlier detection, Data Min. Knowl. Discov. 28 (2014) 190–237.

[23] G. J. Williams, R. A. Baxter, H. X. He, S. Hawkins, L. Gu, A comparative study of rnn for outlier detection in data mining, in: ICDE, 2002, pp. 426–435.

[24] Y. Chen, L. Tu, Density-based clustering for real-time stream data, in: KDD, 2007, pp. 133–142.

[25] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: PODS, 2002, pp. 1–16.

[26] F. Angiulli, F. Fassetti, Detecting distance-based outliers in streams of data, in: CIKM, 2007, pp. 811–820.

[27] D. Yang, E. Rundensteiner, M. Ward, Neighbor-based pattern detection for windows over streaming data, in: EDBT, 2009, pp. 529–540.

[28] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, Y. Manolopoulos, Continuous monitoring of distance-based outliers over data streams, in: ICDE, 2011, pp. 135–146.

[29] K. Yamanishi, J. Takeuchi, A unifying framework for detecting outliers and change points from non-stationary time series data, in: KDD, 2002, pp. 676–681.

[30] S. Budalakoti, A. N. Srivastava, M. E. Otey, Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety, IEEE Transactions on Systems, Man, and Cybernetics, Part C 39 (2009) 101–113.

[31] B. Yi, N. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, A. Biliris, Online data mining for co-evolving time sequences, in: ICDE, 2000, pp. 13–22.

[32] K. Wu, K. Zhang, W. Fan, A. Edwards, P. S. Yu, Rs-forest: A rapid density estimator for streaming anomaly detection, in: ICDM, 2014, pp. 600–609.

[33] C. C. Aggarwal (Ed.), Managing and Mining Sensor Data, Springer, 2013.

[34] B. Wang, X. Yang, G. Wang, G. Yu, Outlier detection over sliding windows for probabilistic data streams, J. Comput. Sci. Technol. 25 (2010) 389–400.

[35] S. A. Shaikh, H. Kitagawa, Efficient distance-based outlier detection on uncertain datasets of gaussian distribution, World Wide Web 17 (2014) 511–538.

[36] J. Zhang, Q. Gao, H. H. Wang, Spot: A system for detecting projected outliers from high-dimensional data streams, in: ICDE, 2008, pp. 1628–1631.

[37] E. J. Keogh, J. Lin, A. W. Fu, HOT SAX: efficiently finding the most unusual time series subsequence, in: ICDM, 2005, pp. 226–233.

[38] V. Niennattrakul, E. J. Keogh, C. A. Ratanamahatana, Data editing techniques to allow the application of distance-based outlier detection to streams, in: ICDM, 2010, pp. 947–952.

[39] S. Papadimitriou, J. Sun, C. Faloutsos, Streaming pattern discovery in multiple time-series, in: VLDB, 2005, pp. 697–708.

[40] I. Assent, P. Kranen, C. Baldauf, T. Seidl, Anyout: Anytime outlier detection on streaming data, in: DASFAA (1), 2012, pp. 228–242.

[41] E. Kushilevitz, N. Nisan, Communication complexity, Cambridge University Press, 1997.

[42] A. A. Razborov, On the distributional complexity of disjointness, Theor. Comput. Sci. 106 (1992) 385–390.

[43] M. Fredman, R. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of the ACM 34 (1987) 596–615.

[44] G. Brodal, Worst-case efficient priority queues, in: SODA, 1996, pp. 52–58.

[45] P. Ciaccia, M. Patella, P. Zezula, M-tree: An efficient access method for similarity search in metric spaces, in: VLDB, 1997, pp. 426–435.

[46] D. Papadias, Y. Tao, F. G., B. Seeger, Progressive skyline computation in database systems, ACM TODS 30 (2005) 41–82.

[47] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu, A framework for clustering evolving data streams, in: VLDB, 2003, pp. 81–92.

[48] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: SDM, 2006.

[49] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, T. Seidl, Moa: Massive online analysis, a framework for stream classification and clustering, Journal of Machine Learning Research - Proceedings Track 11 (2010) 44–50.

[50] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, Y. Manolopoulos, Continuous outlier detection in data streams: an extensible framework and state-of-the-art algorithms, in: SIGMOD, 2013, pp. 1061–1064.