NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing

Susik Yoon and Jae-Gil Lee*
Graduate School of Knowledge Service Engineering, KAIST
Daejeon 34141, South Korea
{susikyoon, jaegil}@kaist.ac.kr

Byung Suk Lee
Department of Computer Science, University of Vermont
Burlington, VT 05405, USA
bslee@uvm.edu

ABSTRACT

This paper addresses the problem of efficiently detecting outliers from a data stream as old data points expire from and new data points enter the window incrementally. The proposed method is based on a newly discovered characteristic of a data stream that the change in the locations of data points in the data space is typically very insignificant. This observation has led to the finding that the existing distance-based outlier detection algorithms perform excessive unnecessary computations that are repetitive and/or canceling out the effects. Thus, in this paper, we propose a novel *set-based* approach to detecting outliers, whereby data points at similar locations are grouped and the detection of outliers or inliers is handled at the group level. Specifically, a new algorithm **NETS** is proposed to achieve a remarkable performance improvement by realizing set-based early identification of outliers or inliers and taking advantage of the "net effect" between expired and new data points. Additionally, NETS is capable of achieving the same efficiency even for a high-dimensional data stream through *two-level dimensional filtering*. Comprehensive experiments using six real-world data streams show 5 to 25 times faster processing time than state-of-the-art algorithms with comparable memory consumption. We assert that NETS opens a new possibility to real-time data stream outlier detection.

PVLDB Reference Format:
Susik Yoon, Jae-Gil Lee, Byung Suk Lee. NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing. *PVLDB*, 12(11): 1303-1315, 2019.
DOI: https://doi.org/10.14778/3342263.3342269

1. INTRODUCTION

1.1 Background and Motivation

Outlier detection is a task to find unusual data points in a given data space [5, 11]. Detecting outliers from a data stream, especially in real-time, is drawing much attention as many applications need to detect anomalies as soon as they occur [2, 4, 14, 15, 21, 22, 23, 24]. The applications range from fraud detection in finance to defect detection in manufacturing to abnormal vital sign detection in healthcare to unusual pattern detection in marketing [3, 10]. All these applications would benefit from identifying those critical events in real-time.

Example 1. Real-time cardiac monitoring is crucial to reducing the morbidity and mortality of patients through early detection and alert of anomalies in heartbeats. Usually, the medical information of a patient is collected from implanted or wearable sensors and transmitted to a server for real-time diagnostics [12]. Outlier detection is evidently a key component of this technology [16], and achieving a low latency is of prime importance to identifying emergency cases like cardiac arrests as quickly as possible. □

For a data stream inherently unbounded, outlier detection is done over a *sliding window* which confines the extent of data within the *most recent* context. A *slide* is a sequence of data points evicted from or added to a window when it moves forward. In the *distance-based* outlier detection mechanism [2, 4, 14, 22, 23] widely adopted for a data stream, an *outlier* is defined as a data point that does not have enough other data points in the vicinity within the current window. Accordingly, as a window slides, expired data points may cause nearby data points to become outliers, and new data points may cause nearby data points to become inliers.

Thus, in most existing algorithms three steps are commonly taken whenever a window slides: (1) expired data points are removed from the window, (2) new data points are added to the window, and (3) outliers are detected from the window. These algorithms maintain an index data structure (e.g., indexed stream buffer [2] and micro-cluster [14]) that supports efficient range search in the window and/or reduces the number of potential outliers. The individual data points that have expired or newly arrived are separately handled using the index [2, 4, 14, 22, 23].

The state-of-the-art algorithms practicing such separate handling of individual data points are missing out a big opportunity to improve the performance of distance-based outlier detection. The opportunity stems from an inherent characteristic of data points that they do *not* vary significantly in their locations in the data space; this characteristic is clearer in a data stream monitored for outlier detection, as by definition outliers occur rarely. In other words, data points in a data stream are likely to be concentrated in a

*Jae-Gil Lee is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342269

Table 1: Concentration ratio of real-world stream data sets. Here, the size refers to the number of data points, and the concentration ratio is explained in Definition 1.

Name	Description	Size	Full (Sub)-dim	Concentration ratio
STK	Stock trading records [19]	1.1M	1	**0.64**
TAO	Oceanographic sensors [18]	0.6M	3	**0.87**
HPC	Electric power consumption [9]	1.0M	7	**0.97**
GAS	Household gas sensors [9]	0.9M	10	**0.88**
EM	Gas sensor array [9]	1.0M	16 (4)	0.42 (**0.90**)
FC	Forest cover types [9]	1.0M	55 (3)	0.44 (**0.66**)

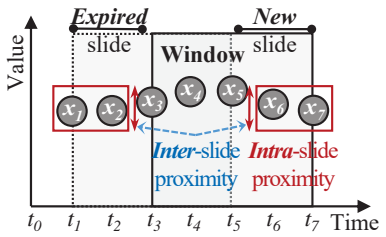
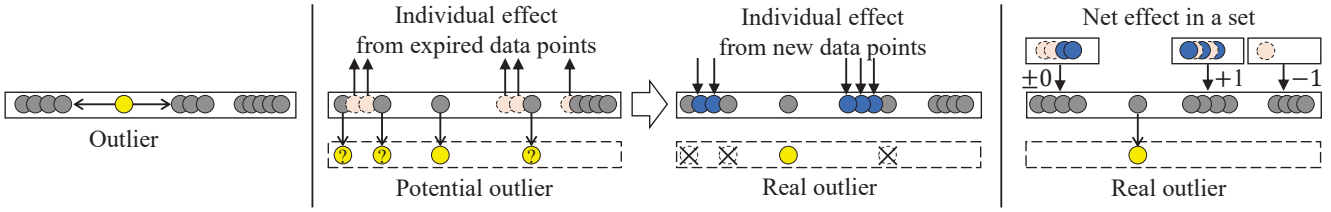


Figure 1: Motivation of using the net effect: x_1 and x_2 are offset (i.e., replaced) by x_6 and x_7.



(a) Previous window. (b) Point-based update. (c) Set-based update.

Figure 2: The two window-update approaches: point-based (b) versus set-based (c). *One*-dimensional data space is considered for ease of exposition.

set of small regions in the data space. Table 1 confirms the characteristic observed from six real-world stream data sets frequently cited in the literature [22]. All of them show a very high "concentration ratio" (see Definition 1) in their full- or sub-dimensional spaces.

Definition 1. (CONCENTRATION RATIO) The *concentration ratio* of a data set is an indicator of how concentrated the data points are in the data space partitioned into hypercubes, called *cells*, of the same size, and is calculated as the ratio of the total number of data points in the top quarter of most populated cells to the number of data points in the entire data space. (This definition has been adapted from the same term used in the field of economics [17].) □

When the concentration ratio is high, as illustrated in Figure 1, data points in the same slide (either expired or new) tend to be close to one another, and likewise, data points in the expired slide are close to data points in the new slide. We call the former the *intra-slide* proximity and the latter the *inter-slide* proximity. The existing algorithms do not recognize the intra-slide proximity and, therefore, repeatedly perform similar updates of an index for expired or new data points, consequently wasting much computing time and memory. Even worse, many data points removed from the expired slide are likely to be *offset* (i.e., replaced) by data points added to the new slide as the data points in the two slides are close to each other. The existing algorithms do not recognize the inter-slide proximity, either, and therefore unnecessarily perform updates for the expired slide, only to be countered by updates for the new slide.

1.2 Main Contributions and Summary

In this paper, we propose an innovative approach based on *set-based update* to make distance-based outlier detection from data streams extremely fast by fully taking advantage of both intra-slide and inter-slide proximity. Close data points in a slide are grouped into a set to avoid repetition for the individual data points. Moreover, this set-based processing makes it possible to extract the net change between expired and new data points in each set, thereby avoiding repetitive and useless update operations when the window slides. Reiterating this idea from the perspective of the algorithmic procedure, when a window slides, the existing algorithms process the expired and new slides *separately* and *sequentially*[1], whereas our approach processes the two slides *together* and *concurrently*. To highlight the difference from our set-based update approach, we call the existing approach the *point-based* update approach.

Key idea: Figure 2 illustrates how the set-based update approach is different from the point-based update approach. Consider the previous window in Figure 2a that has just slid, where there is a single outlier (indicated in yellow). The point-based update in Figure 2b performs update separately and sequentially as the individual expired data points are removed from the current window, as indicated with the upward arrows. Three data points lose their neighbors and thus become potential outliers. Immediately following it, however, the point-based update performs another update separately and sequentially as the individual new data points are added to the current window, as indicated with the downward arrows. The three potential outliers obtain enough neighbors and thus become inliers. Note that there is no change of outliers after the window slides. In contrast, the set-based update in Figure 2c first compares close data points in the expired and new slides and handles them together and concurrently as sets, as indicated with the small rectangles. Then, it calculates the net effect in each set (i.e., ± 0 for no change, $+1$ for one net new data point, or -1 for one net expired data point) as reflected in the current window. As a result of this set-based update, (1) the number of updates performed is much smaller, and (2) no potential outlier is generated *unnecessarily*, i.e., only to be canceled out immediately when new data points are looked at.

Algorithm: We propose a novel algorithm, called **NET-effect-based Stream outlier detection (NETS)**, that performs distance-based outlier detection from a data stream while fully capitalizing on the key idea discussed

[1]The order of processing the two slides is immaterial.

above. To the best of our knowledge, this algorithm is the first and foremost in realizing the set-based update and combining updates from both expired and new slides. Concretely, the following two techniques are employed.

- *Grid index*: For the set-based update, it is very important to efficiently group close data points in a slide. To this end, NETS uses a single-level grid index where all grid cells have the same size. Each cell is essentially the implementation of a set. The size of a cell is determined by a parameter typical of distance-based outlier detection. The set (i.e., cell) to which a data point belongs is identified very quickly using the grid index, and thus the cardinality of a set is maintained easily and efficiently. Moreover, the net effect of changes from the two slides (i.e., expired and new) is calculated at a small cost equivalent to that of matrix addition.
- *Two-level dimensional filtering*: The concentration ratio is usually low in the full-dimensional space of a high-dimensional data stream, as shown in Table 1. So, to support high dimensionality as efficiently as low dimensionality, NETS first selects a subset of dimensions, such that the concentration ratio is sufficiently high, in order to detect outliers *early on* (i.e., at the cell level). Then, only the data points of which the outlier-ness cannot be determined in the sub-dimensional space are processed further in the full-dimensional space. Consequently, only a small portion of the full-dimensional data space is looked at. We have developed a heuristic algorithm that can select an "optimal" subset of dimensions to maximize the benefit of the two-level dimensional filtering.

Performance improvement: The key idea of NETS is fairly straightforward, and yet its impact on the performance improvement is outstanding. We show it through comprehensive experiments done using one synthetic data set and six real-world data sets commonly cited in the literature. MCOD [14] has been known to be the best performer among the existing algorithms [22]. As will be shown in Section 6, NETS outperforms MCOD by 5–150 times while consuming only comparable memory space.

The rest of the paper is organized as follows. Section 2 defines the problem formally. Section 3 reviews the state-of-the-art related work. Section 4 discusses the NETS algorithm in detail. Section 5 elaborates on the two-level dimensional filtering. Section 6 presents the results of experiments. Section 7 concludes the paper.

2. PRELIMINARY (PROBLEM SETTING)

This section provides the formal definition of the problem of *Distance-based Outlier Detection in Data Streams (DODDS)*. The notations used throughout this paper are summarized in Table 2.

First, *distance-based outliers* for static data are defined in Definitions 2 and 3.

Definition 2. (NEIGHBOR) Given a distance threshold θ_R, a data point x_i is a *neighbor* of another data point x_j ($x_i \neq x_j$) if the distance between x_i and x_j is no more than θ_R. □

Definition 3. (DISTANCE-BASED OUTLIER/INLIER) Given a set X of data points, a neighbor count threshold θ_K, and a distance threshold θ_R, a data point x in X is a *distance-based outlier* if x has fewer than θ_K neighbors in X and, otherwise, a *distance-based inlier*. □

Table 2: Summary of the notations in the paper.

Notation	Description
$\mathcal{D}^d$	a d-dimensional domain space
D_{full}	the set of full-dimensions in $\mathcal{D}^d$
D_{sub}	a set of sub-dimensions in $\mathcal{D}^d$
x	a data point
W	a window
S_{new}	a new slide of a window
S_{exp}	an expired slide of a window
c	a cell
O	a set of outliers
θ_W	the size of a window
θ_S	the size of a window slide
θ_R	the threshold on the distance of a neighborhood
θ_K	the threshold on the number of neighbors

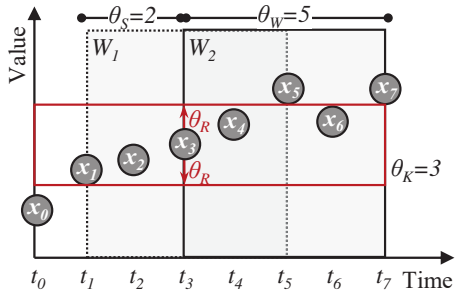


Figure 3: Example of DODDS. A data point x_3 was an inlier in W_1, but it becomes an outlier in W_2.

Then, a *data stream* and a related window concept are given in Definitions 4 and 5.

Definition 4. (DATA STREAM) A *data stream* is an infinite sequence of data points, $\ldots, x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, \ldots$, arriving in an increasing order of the timestamp. □

As new data points arrive continuously, data streams are usually processed using a *sliding window* containing the set of most recent data points. In this paper, a sliding window is formalized as the *count-based* window of Definition 5, as in a majority of other studies [2, 4, 14, 22, 23]. Hereafter, a count-based window is referred simply by a *window*.

Definition 5. (COUNT-BASED WINDOW) A *count-based window* W of size θ_W at a data point x_i is the set of data points, $\{x_{i-\theta_W+1}, x_{i-\theta_W+2}, \ldots, x_i\}$. □

A window moves by θ_S data points at once; thus, each time a window moves, θ_S old data points are removed, and θ_S new data points are added. The removed portion is called an *expired slide*, and the added portion is called a *new slide*.

Finally, the framework of the paper—*DODDS*—is formulated by Definition 6.

Definition 6. (DODDS) *Distance-based outlier detection in data streams (DODDS)* is a problem of finding a set of outliers according to Definition 3 in every window of size θ_W sliding at the increment of θ_S, provided with a neighbor count threshold θ_K and a distance threshold θ_R. □

Example 2. Figure 3 illustrates an example scenario of DODDS. Suppose that $\theta_W = 5$, $\theta_S = 2$, and $\theta_K = 3$. In W_1, a data point x_3 has three neighbors, x_1, x_2, and x_4, within θ_R and hence is regarded as an inlier. However, in W_2, x_3 becomes an outlier as it loses two old neighbors, x_1 and x_2, and acquires only one new neighbor, x_6. □

Hereafter, we omit "distance-based" in this paper when discussing outlier detection.

Table 3: Categorization of the DODDS algorithms.

		Maintain window by	
		Existing index	Custom index
Identify outlier by	Neighbor list	Abstract-C [23]	Exact/approx-Storm [2]
	Potential outlier	DUE [14] LEAP [4]	MCOD [14]
	Net effect		**NETS**

3. RELATED WORK

Existing DODDS algorithms are discussed in this section. The recent survey by Tran et al. [22] provides a comprehensive summary and comparison of six representative algorithms: exact-Storm [2], Abstract-C [23], DUE [14], MCOD [14], LEAP [4], and approx-Storm [2]. Given the typically high-speed streaming nature of data streams, all these algorithms update outliers *incrementally* every time the window slides by (1) removing old data points in an expired slide, (2) adding new data points in a new slide, and (3) finding outliers from the "active" data points. To this end, their focus has been on efficiently maintaining the current window and/or efficiently identifying outliers. The techniques used for the former include existing indexes such as M-tree [7] or custom indexes; those for the latter include neighbor lists or potential outliers. The existing algorithms and NETS can be categorized by their design, as shown in Table 3.

In exact-Storm [2], each data point x is associated with a list of up to θ_K *preceding neighbors* (i.e., expired before x) and a number of *succeeding neighbors* (i.e., not expired before x). This neighbor information is enough to determine whether each data point is an outlier. Abstract-C [23] maintains a summary of the neighbor information in a different way, which manages the *lifetime neighbor count* of each data point in every window that it belongs to. Because a new neighbor will remain in a constant (i.e., θ_W/θ_S) number of windows, the maximum length of the list is θ_W/θ_S. Outliers can be identified by simply referring to the current window's neighbor count of each data point. While exact-Storm and Abstract-C try to optimize neighbor lists, DUE [14] focuses on managing potential outliers, which are likely to become outliers by losing expired neighbors. The inliers are stored in a data structure, called an *event queue*, in the increasing order of the earliest expiration time of their neighbors. In every window, the event queue is used to re-evaluate only the inliers whose neighbors have just expired.

State-of-the-art algorithms: MCOD and LEAP consistently outperformed the other algorithms [22], and so let us discuss the two algorithms in detail.

MCOD [14] uses an index structure called a *micro-cluster* to efficiently prune out unqualified outlier candidates. If there are more than θ_K data points inside a circle with a radius of $\theta_R/2$, a micro-cluster is formed to guarantee that all members are inliers. Some inliers not in a micro-cluster are managed in an *event queue* similarly to DUE. In every window, the update works in three steps: **(1)** *processing an expired slide*, where (i) if the size of a micro-cluster containing expired data points falls below $\theta_K + 1$, its members are regarded as new data points, and (ii) the data points which had the expired data points as neighbors are retrieved from the event queue and checked for their outlier status; **(2)** *processing a new slide*, where new data points are attempted to either join the closest micro-cluster or form a new one, and if both attempts fail, then the data points are entered into the event queue; **(3)** *identifying outliers*, where data points that are not in any micro-cluster and have fewer than θ_K neighbors become outliers.

LEAP [4] suggests a *minimal probing principle* to find only the minimum number of neighbors prioritized by their arriving time, using indexes built per slide. It uses a *trigger list* to manage the data points affected by expired data points. In every window, the update works as follows: **(1)** *processing a new slide*, where (i) neighbors of new data points are searched by probing the new slide and then the preceding slides in reverse chronological order until finding θ_K neighbors, and (ii) the new data points are added to the trigger list of each probed slide; **(2)** *processing an expired slide*, where (i) the data points in the trigger list of the expired slide are re-evaluated by checking the number of neighbors found, and (ii) if each data point has fewer than θ_K neighbors, more neighbors are searched from the succeeding slides in chronological order; **(3)** *identifying outliers*, where data points having fewer than θ_K neighbors become outliers.

The superiority of MCOD and LEAP over exact-Storm, Abstact-C, and DUE is mainly due to the reduction in the frequency of range searches for finding neighbors because of micro-clusters (MCOD) and slide-based indexes (LEAP).

Limitations of the existing algorithms: Unlike NETS, these algorithms do not exploit the net effect between expired and new data points and, consequently, are not able to avoid redundant updates when a window slides. Moreover, many potential outliers identified because of expired neighbors quickly revert to inliers because of new neighbors.

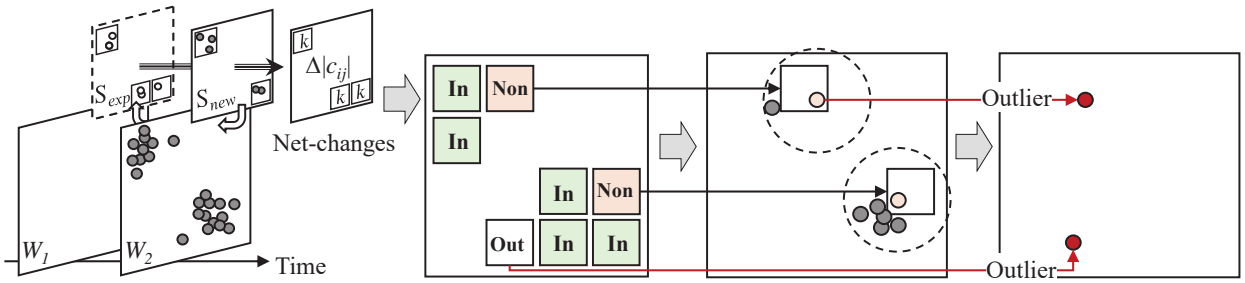
4. THE ALGORITHM "NETS"

4.1 Overview

The overall procedure of the NETS algorithm is outlined in Algorithm 1 and illustrated in Figure 4. As preprocessing, NETS finds the optimal set of dimensions that minimizes the outlier detection cost estimated based on the concentration ratio observed in a sample of the data (Line 1). This optimal set may include the entire set of dimensions, especially for a low-dimensional data stream. As the data stream arrives continuously, for each window on the stream, outliers are detected from the current window through *cell-level* detection and then *point-level* detection (Lines 2–25).

Cell-level detection (Lines 5–21): If all data points in a specific cell have the same outlier status (either all outliers or all inliers), they can be identified as outliers or inliers early at this level. This cell-level detection is done through *two-level dimensional filtering*, i.e., sub-dimensional (Lines 6–13) followed by full-dimensional (Lines 14–21), provided that the optimal set of dimensions (D_{sub}) is a proper subset of the set of all dimensions (D_{full}). The filtering procedure is the same between the sub-dimensional filtering and the full-dimensional filtering. Only those cells that do not qualify for early detection in the sub-dimensional space are deferred to the full-dimensional filtering, and it effectively reduces the full-dimensional search space.

NETS maintains a *cardinality grid* ($\mathbb{G}_{sub}$ and $\mathbb{G}_{full}$), where the cardinality of data points is stored in each grid cell. The cardinality values calculated from the previous window are retrieved for update (Lines 8, 15). Next, NETS calculates the net effect (Δ_{sub} and Δ_{full}) between expired



(a) Slide net effect calculation. (b) Cell-level detection. (c) Point-level detection. (d) Final outliers.

Figure 4: Overall procedure of our proposed algorithm NETS (when $D_{sub} = D_{full}$).

Algorithm 1 The Overall Procedure of NETS

INPUT: a data stream S, a set D_{full} of dimensions;
OUTPUT: a set O of outliers for each sliding window;
1: $D_{sub} \leftarrow$ GetOptDimensions(S); /* preprocessing */
2: **for each** window W from S **do**
3: Let S_{exp} be the expired slide and S_{new} be the new slide;
4: $O \leftarrow \emptyset$; /* outliers in the current window */
5: /* CELL-LEVEL DETECTION */
6: **if** $D_{sub} \subset D_{full}$ **then**
7: /* SUB-DIMENSIONAL FILTERING */
8: $\mathbb{G}_{sub} \leftarrow$ cardinality grid in D_{sub};
9: $\Delta_{sub} \leftarrow$ CalcNetEffect(S_{exp}, S_{new}, D_{sub});
10: $\mathbb{G}_{sub} \leftarrow \mathbb{G}_{sub} + \Delta_{sub}$;
11: $(\emptyset, \mathbb{C}_{sub}^{outlier}, \mathbb{C}_{sub}^{non}) \leftarrow$ CategorizeCells($\mathbb{G}_{sub}$);
12: $O \leftarrow O \cup \{x \mid x \in \mathbb{C}_{sub}^{outlier}\}$;
13: $\mathbb{G}_{full} \leftarrow$ cardinality grid in $D_{full} \cap \mathbb{C}_{sub}^{non}$;
14: **else**
15: $\mathbb{G}_{full} \leftarrow$ cardinality grid in D_{full};
16: **end if**
17: /* FULL-DIMENSIONAL FILTERING */
18: $\Delta_{full} \leftarrow$ CalcNetEffect(S_{exp}, S_{new}, D_{full});
19: $\mathbb{G}_{full} \leftarrow \mathbb{G}_{full} + \Delta_{full}$;
20: $(\mathbb{C}_{full}^{inlier}, \mathbb{C}_{full}^{outlier}, \mathbb{C}_{full}^{non}) \leftarrow$ CategorizeCells($\mathbb{G}_{full}$);
21: $O \leftarrow O \cup \{x \mid x \in \mathbb{C}_{full}^{outlier}\}$;
22: /* POINT-LEVEL DETECTION */
23: $O \leftarrow O \cup$ FindOutlierPoints($\mathbb{C}_{full}^{non}$, $\mathbb{G}_{full}$);
24: **return** O;
25: **end for**

and new data points (Lines 9, 18) (see Figure 4a) and then applies the net effect to the cardinality grid and detects outliers at the cell level using the updated cardinality values (Lines 10–12, 19–21) (see Figure 4b). This way, each non-empty cell is categorized into one of outlier, inlier, and non-determined cells, as shown in Figure 4b.

Point-level detection (Lines 22–23): NETS inspects the data points in non-determined cells further at the point level (Line 23) (see Figure 4c). Then, the outliers detected at both cell- and point-levels are returned as the final output (see Figure 4d).

4.2 Cardinality Grid

NETS uses a *cardinality grid* $\mathbb{G}_D$, a cell-based structure in a multi-dimensional space D, to maintain the net effect in each cell. The data space D is partitioned into hypercubes, called "D-cells." For a given window or slide, the *cardinality* of a D-cell c, which is denoted as $card(c)$, indicates the number of data points within c. The cardinality grid stores the cardinality for each D-cell.

NETS limits the diagonal length of every D-cell to θ_R (i.e., side length to $\theta_R/\sqrt{|D|}$), thereby enforcing the distance between any two data points in a D-cell to be no longer than θ_R. This specification leads to Lemma 1.

LEMMA 1. *All data points in a D-cell are neighbors with one another in the data space D.*

PROOF. Since the maximum distance possible between two points in a D-cell is θ_R, any two data points x_i and x_j in a D-cell are neighbors by Definition 2. □

We further define the neighbor condition between D-cells in Definition 7.

Definition 7. (NEIGHBOR CELL) Two different D-cells are said to be *neighbors* in D if the distance between the centers of the two D-cells is no longer than $2\theta_R$. □

LEMMA 2. *Let us consider two data points x_i and x_j in two different D-cells c_i and c_j, respectively. The centers of c_i and c_j are denoted as $c_i.ctr$ and $c_j.ctr$. If $dist(x_i, x_j) \leq \theta_R$, then $dist(c_i.ctr, c_j.ctr) \leq 2\theta_R$, where $dist(\cdot,\cdot)$ is the Euclidean distance.*

PROOF. The center of a D-cell is at the center of its diagonal line whose length is θ_R, so the distance between a data point and the center is at most $\theta_R/2$. Then, by the triangle inequality, $\text{dist}(c_i.ctr, c_j.ctr) \leq \text{dist}(c_i.ctr, x_i) + \text{dist}(x_i, x_j) + \text{dist}(x_j, c_j.ctr) \leq \theta_R/2 + \theta_R + \theta_R/2 = 2\theta_R$. □

Let $N(c)$ denote the set of all neighbors of a D-cell c. Then, the implication of Lemma 2 is that, for any data point x in c, the set of data points in $N(c)$ is a *superset* of the neighbors of x that are located outside c.

4.3 Step 1: Slide Net-Effect Calculation

NETS calculates the net effect of expired and new slides using the cell-based cardinality grid data structure. By Lemma 1, the net change in the number of neighbors within a D-cell applies equally to all data points in the D-cell. This property enables NETS to quickly identify outliers in both the cell-level and point-level detection steps.

Algorithm 2 outlines the procedure. Each data point in the expired slide is added to the cardinality grid of expired data points ($\mathbb{G}_{exp}$) in the D-cell containing the coordinate of the data point (Lines 1–3), and likewise each data point in the new slide is added to that of new data points ($\mathbb{G}_{new}$) (Lines 4–6). Here, the grid cell for a data point can be easily calculated (in constant time) by dividing the coordinate value of the data point in each dimension by the side

Algorithm 2 CalcNetEffect()

INPUT: S_{exp}, S_{new}, and D;
OUTPUT: Net effect Δ in each cell of data space D;
1: **for** each data point x in S_{exp} **do**
2: Add x to the $\mathbb{G}_{exp}$ cell containing the coordinate of x;
3: **end for**
4: **for** each data point x in S_{new} **do**
5: Add x to the $\mathbb{G}_{new}$ cell containing the coordinate of x;
6: **end for**
7: $\Delta \leftarrow \mathbb{G}_{new} - \mathbb{G}_{exp}$; /* net effect */
8: **return** Δ;

length of the D-cell. After processing all data points in the two slides, the net effect Δ is calculated by subtracting the cardinality of expired data points from the cardinality of new data points in each D-cell (Line 7). This operation was implemented using matrix *addition*, which is computed in linear time with the number of non-empty cells.

4.4 Step 2: Cell-Level Outlier Detection

NETS tries to detect outliers and inliers early at the cell-level by finding D-cells whose data points are either all outliers or all inliers. To this end, we can derive lower- and upper-bounds on the number of neighbors of a data point in a D-cell as stated in Lemmas 3 and 4.

LEMMA 3. *For a given D-cell c, the* lower-bound $L(c)$ *in D_{full} on the number of neighbors for any data point x in c is equal to $card(c) - 1$ if $D = D_{full}$, and unknown if $D \subset D_{full}$.*

PROOF. The number of neighbors of x inside c is exactly $card(c) - 1$ (excluding x) by Lemma 1. This count is the lower bound $L(c)$ in D_{full}. On the other hand, it does not hold if $D \subset D_{full}$ since not all neighbors in D_{sub} are also neighbors in D_{full}. □

LEMMA 4. *For a given D-cell c, the* upper-bound $U(c)$ *in D_{full} on the number of neighbors for any data point x in c is equal to $\sum_{c' \in N(c)} card(c') + card(c) - 1$, where $N(c)$ is the set of neighbor cells of c.*

PROOF. The number of neighbors of x inside c is exactly $card(c) - 1$ (excluding x) by Lemma 1, and that of x outside c is at most $\sum_{c' \in N(c)} card(c')$ by Lemma 2. Thus, the sum of these two counts becomes the upper bound $U(c)$. This bound holds even when $D \subset D_{full}$ since neighbors in D_{sub} include all neighbors in D_{full}. □

Given these two bounds, a D-cell is categorized into one of three types: an *inlier cell*, an *outlier cell*, and a *non-determined cell* as defined in Definitions 8 through 10.

Definition 8. (INLIER CELL) A D-cell is called an *inlier cell* if all data points in the D-cell are inliers. □

Definition 9. (OUTLIER CELL) A D-cell is called an *outlier cell* if all data points in the D-cell are outliers. □

Definition 10. (NON-DETERMINED CELL) A D-cell is called a *non-determined cell* if it is neither an inlier cell nor an outlier cell. □

Theorems 1 and 2 state the criteria used to determine the correct cell type.

THEOREM 1. *A D-cell c that satisfies $L(c) \geq \theta_K$ is an inlier cell.*

Algorithm 3 CategorizeCells()

INPUT: a cardinality grid $\mathbb{G}$;
OUTPUT: $\mathbb{C}_{inlier}, \mathbb{C}_{outlier}$, and $\mathbb{C}_{non}$;
1: $\mathbb{C}_{inlier}, \mathbb{C}_{outlier}, \mathbb{C}_{non} \leftarrow \emptyset$;
2: **for each** cell $c \in \mathbb{G}$ **do**
3: **if** $L(c) \geq \theta_K$ **then**
4: $\mathbb{C}_{inlier} \leftarrow \mathbb{C}_{inlier} \cup \{c\}$; /* Theorem 1 */
5: **else if** $U(c) < \theta_K$ **then**
6: $\mathbb{C}_{outlier} \leftarrow \mathbb{C}_{outlier} \cup \{c\}$; /* Theorem 2 */
7: **else**
8: $\mathbb{C}_{non} \leftarrow \mathbb{C}_{non} \cup \{c\}$;
9: **end if**
10: **end for**
11: **return** $\mathbb{C}_{inlier}, \mathbb{C}_{outlier}, \mathbb{C}_{non}$;

PROOF. For every data point x in the D-cell c, x is an inlier because the number $\mathcal{N}(x)$ of neighbors satisfies $\mathcal{N}(x) \geq L(c) \geq \theta_K$. Therefore, by Definition 8, the D-cell c is an inlier cell. □

THEOREM 2. *A D-cell c that satisfies $U(c) < \theta_K$ is an outlier cell.*

PROOF. For every data point x in the D-cell c, x is an outlier because the number $\mathcal{N}(x)$ of neighbors satisfies $\mathcal{N}(x) \leq U(c) < \theta_K$. Therefore, by Definition 9, the D-cell c is an outlier cell. □

Algorithm 3 outlines the procedure of cell-level outlier detection. For each D-cell in the given cardinality grid $\mathbb{G}$, the algorithm calculates the lower- and upper-bounds and determines the type of the D-cell according to its bounds (Lines 3–9). The algorithm returns the three types of sets: $\mathbb{C}_{inlier}$ for inlier cells, $\mathbb{C}_{outlier}$ for outlier cells, and $\mathbb{C}_{non}$ for non-determined cells (Line 11), while only $\mathbb{C}_{non}$ is passed to the next step—point-level outlier detection.

4.5 Step 3: Point-Level Outlier Detection

NETS detects outliers at the point level by inspecting each data point in non-determined cells. NETS attempts to exploit the cell-level information in this step as well to the extent possible. Interestingly, still some data points in non-determined cells can be quickly identified as inliers by using the cardinality grid and the number of neighbors already known from the previous window. To make it possible, NETS distinguishes the neighbors of a data point as stated in Definition 11 depending on whether they are located in the same D-cell as the data point or not.

Definition 11. (INNER OR OUTER NEIGHBORS) Given a data point x in a D-cell c, the *inner neighbors* of x are the neighbors inside c, and the *outer neighbors* of x are the neighbors outside c. Their counts are denoted as $\mathcal{N}^{in}(x)$ and $\mathcal{N}^{out}(x)$, respectively. □

These two types of neighbor counts are implemented as follows. For a data point x, while $\mathcal{N}^{in}(x)$ can be obtained exactly from the cardinality grid of the current window, $\mathcal{N}^{out}(x)$ requires examining individual data points in all neighbor cells. NETS reduces this cost by counting outer neighbors *conservatively*. Specifically, as NETS examines each slide in a window, it stores the counts of outer neighbors per slide until the cumulative count reaches θ_K (as done by LEAP [4]). Then, in the next window, the sum of those counts over the slides except the expired one is used as a conservative count of outer neighbors. The data point x is guaranteed to be an inlier in the new window provided with the condition in Theorem 3, as illustrated in Example 3.

Algorithm 4 FindOutlierPoints()

INPUT: $\mathbb{C}_{non}$ and $\mathbb{G}$;
OUTPUT: a set O of outliers;
1: $O \leftarrow \emptyset$;
2: **for each** cell $c \in \mathbb{C}_{non}$ **do**
3: **for each** data point $x \in c$ **do**
4: $\mathcal{N}^{in}(x) \leftarrow card(c)$ in $\mathbb{G}$; /* exact count */
5: $\mathcal{N}^{out}(x) \leftarrow$ conservative count from the previous window (not including the new slide);
6: **if** $\mathcal{N}^{in}(x) + \mathcal{N}^{out}(x) < \theta_K$ **then**
7: /* Find more neighbors at the point level */
8: **do** range search in $N(c)$ to update $\mathcal{N}^{out}(x)$ **until** $\mathcal{N}^{out}(x) \geq \theta_K - \mathcal{N}^{in}(x)$;
9: **if** $\mathcal{N}^{out}(x) < \theta_K - \mathcal{N}^{in}(x)$ **then**
10: $O \leftarrow O \cup \{x\}$; /* not enough neighbors */
11: **end if**
12: **end if**
13: **end for**
14: **end for**
15: **return** O;

THEOREM 3. *Given the exact $\mathcal{N}^{in}(x)$ and the conservative $\mathcal{N}^{out}(x)$, a data point x is an inlier if $\mathcal{N}^{in}(x) + \mathcal{N}^{out}(x) \geq \theta_K$.*

PROOF. Since $\mathcal{N}^{out}(x)$ is a conservative count of outer neighbors, there may be other outer neighbors in the unexamined slides (including the new slide), and thus the exact number of neighbors $\mathcal{N}(x) \geq \mathcal{N}^{in}(x) + \mathcal{N}^{out}(x)$. So, x is an inlier because $\mathcal{N}(x) \geq \theta_K$ by transitivity. □

Example 3. Consider a window W_1 composed of four slides S_1, S_2, S_3, and S_4, shown in Figure 5. Let θ_K be 5. Suppose that a data point x in S_2 has $\mathcal{N}^{in}(x) = 2$ in W_1 and that NETS finds one outer neighbor of x in S_1 and two outer neighbors of x in S_2. Then, $\mathcal{N}^{out}(x) = 3$. Since the total number of neighbors found is $5\,(= 2 + 3) \geq \theta_K$, NETS stops examining the other slides and classifies x as an inlier. Now consider the next window W_2, composed of S_2, S_3, S_4, and S_5, and suppose $\mathcal{N}^{in}(x) = 3$. The conservative $\mathcal{N}^{out}(x)$ in W_2 is reduced to 2 as the count from S_1 is subtracted. Nevertheless, the total number of neighbors found is still $5\,(= 3 + 2) \geq \theta_K$, and, therefore, x is still identified as an inlier without examining the rest of the window. □

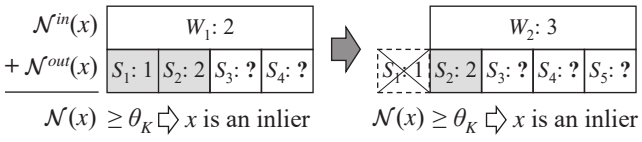


Figure 5: An illustration of Example 3 for a data point x located in the slide S_2.

Algorithm 4 outlines the procedure of point-level outlier detection. For each data point in each non-determined cell, $\mathcal{N}^{in}(x)$ is updated using the cardinality grid $\mathbb{G}$ (Line 4), and $\mathcal{N}^{out}(x)$ is estimated from the result of the previous window (Line 5). If such a data point is not guaranteed to be an inlier, the algorithm updates $\mathcal{N}^{out}(x)$ further by probing each of the neighbor cells $N(c)$ in the unexamined slides (Lines 6–8). If there are still fewer than θ_K neighbors, the data point is classified as an outlier (Lines 9–10). Finally, NETS returns the set of outliers (Line 15).

4.6 Complexity Analysis

The worst-case time and space complexities of NETS are given by Theorems 4 and 5 respectively.

THEOREM 4. *Given the number N_C of non-empty D-cells in a window, the time complexity of NETS is $O(\theta_S + N_C\theta_W)$.*

PROOF. The time complexity of calculating the net effect is $O(\theta_S + N_C)$. The time complexity of cell-level detection is $O(N_C)$. The time complexity of point-level detection is $O(N_C\theta_W\theta_K)$ because the number of data points in non-determined cells is at most $N_C\theta_K$, and the number of their neighbor candidates is at most θ_W. In practice, because θ_K/θ_W is almost 0, $O(N_C\theta_W\theta_K)$ can be approximated as $O(N_C\theta_W)$ [13]. Thus, the overall time complexity is $O(\theta_S + N_C + N_C\theta_W) = O(\theta_S + N_C\theta_W)$. □

When a data stream has a high concentration ratio, the number of non-empty D-cells becomes very small, and the time complexity of NETS can be further reduced to $O(\theta_S + \theta_W)$. According to the survey by Tran et al. [22], the time complexity of MCOD is $O((1-c)\theta_W \log((1-c)\theta_W) + \theta_K\theta_W \log\theta_K)$, where c is the proportion of the data points in micro-clusters, and the time complexity of LEAP is $O(\theta_W^2 \log\theta_W/\theta_S)$. Therefore, NETS has a lower time complexity than MCOD and LEAP.

THEOREM 5. *Given the number N_C of non-empty D-cells in a window, the space complexity of NETS is $O(N_C + \theta_W^2/\theta_S)$.*

PROOF. The space complexity of managing D-cells is $O(N_C)$, and that of keeping neighbor counts is $O(\theta_W^2/\theta_S)$ because each of θ_W data points stores the count for each of θ_W/θ_S slides [4]. Thus, the space complexity of NETS is $O(N_C + \theta_W^2/\theta_S)$. □

According to the survey [22] again, the space complexity of MCOD is $O(c\theta_W + (1-c)\theta_K\theta_W)$, and that of LEAP is $O(\theta_W^2/\theta_S)$. Thus, NETS has the same space complexity as LEAP and also has similar complexity to MCOD in practice because typically θ_W/θ_S is constant.

5. HIGH DIMENSIONALITY SUPPORT

5.1 Sub-Dimensional Filtering

Data points in a high-dimensional data set are usually sparsely distributed, commonly known as "the curse of high dimensionality." To NETS, it means that there are fewer data points in D_{full}-cells, which diminishes the performance advantage of set-based processing. To overcome this issue, NETS first processes D_{sub}-cells to discover outliers and inliers in a sub-dimensional space that does not suffer from the data sparsity and then processes the remaining D_{sub}-cells further in a full-dimensional space. This two-level dimensional filtering is justified by the *downward closure property* in Lemma 5.[2]

LEMMA 5. *(*DOWNWARD CLOSURE PROPERTY OF NEIGHBORS*) Data points neighboring in a full-dimensional space D_{full} are neighbors in any sub-dimensional space $D_{sub} \subseteq D_{full}$ as well.*

PROOF. The Euclidean distance between two data points in D_{full} must be greater than or equal to that in D_{sub} by definition. Therefore, the neighbors in D_{sub} is always inclusive of those in D_{full}. □

[2]This property has been first introduced in frequent pattern mining [1] and is also used in subspace clustering [6, 20].

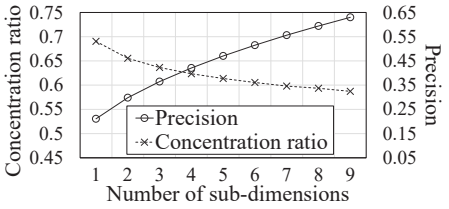


Figure 6: Concentration ratio and precision for varying sub-dimensionality (tested on a 10-dimensional synthetic data set generated from a Gaussian mixture model).

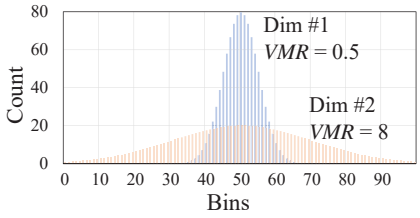


Figure 7: VMRs of two selected dimensions from the data set in Figure 6.

By Lemma 5, data points in a neighboring D_{sub}-cell contain all data points in neighboring D_{full}-cells that are reduced to the D_{sub}-cell. Thus, by examining D_{sub}-cells that have a high concentration ratio, NETS can identify outlier D_{sub}-cells earlier in the cell-level detection step and, additionally, reduce the search space of non-determined D_{full}-cells in the point-level detection step. A high concentration ratio, on the other hand, is not always beneficial, since the converse of Lemma 5 is not always true. That is, a higher concentration ratio in D_{sub} is likely to cause more neighbors that later prove to be false neighbors in D_{full}. This situation lowers the precision, i.e., the ratio of true neighbors in D_{full} over all neighbors identified in D_{sub}. Having to purge out these false positive neighbors in D_{full} may incur a nontrivial overhead. Since concentration ratio is negatively correlated with sparsity and sparsity is positively correlated with the dimensionality of D_{sub}, there exists a trade-off between concentration ratio and precision depending on the sub-dimensionality, as shown in Figure 6.

5.2 Optimal Sub-Dimensionality Selection

The exponential number of possible sub-dimensional spaces warrants an efficient mechanism to find an optimal sub-dimensional space balancing the trade-off between concentration ratio and precision. NETS uses a systematic approach based on a prioritization of individual dimensions and the associated cost of detecting outliers.

Priority of a dimension: The well-known index of dispersion in time series data, *variance-to-mean ratio (VMR)* [8] in Definition 12, is used as the priority.

Definition 12. [8] (VARIANCE-TO-MEAN RATIO) Given a set of data points, its *variance-to-mean ratio* in the k-th dimension, VMR_k, is defined as σ_k^2/μ_k, where σ_k^2 and μ_k are, respectively, the variance and the mean of the distribution of the data points in the k-th dimension. □

A dimension with a lower VMR is less dispersed in the data distribution and shows a higher concentration ratio; therefore, it is given a higher priority in the sub-dimensionality selection. For an illustration using Figure 7, the first dimension (Dim #1) with the lower VMR is less dispersed than the second dimension (Dim #2) and, therefore, has the higher priority.

Algorithm 5 GetOptDimensions()

INPUT: a data stream S
OUTPUT: optimal sub-dimensions D
1: $X \leftarrow$ sample data points from S;
2: Calculate VMR of each dimension in full dimensional space D_{full} and sort the dimensions in increasing order of VMR;
3: Calculate the cost $C_{D_{full}}$ of detecting outliers from X in D_{full} (using Eq. (1) for $D_{sub} = D_{full}$);
4: $D_{sub} \leftarrow \emptyset$;
5: **repeat**
6: Add to D_{sub} a new dimension in D_{full} that maximizes the reduction in the outlier detection cost $C_{D_{sub}}$ (calculated using Eq. (1));
7: **until** $C_{D_{sub}}$ is not decreased;
8: **if** $C_{D_{sub}} < C_{D_{full}}$ **then**
9: **return** D_{sub};
10: **else**
11: **return** D_{full};
12: **end if**

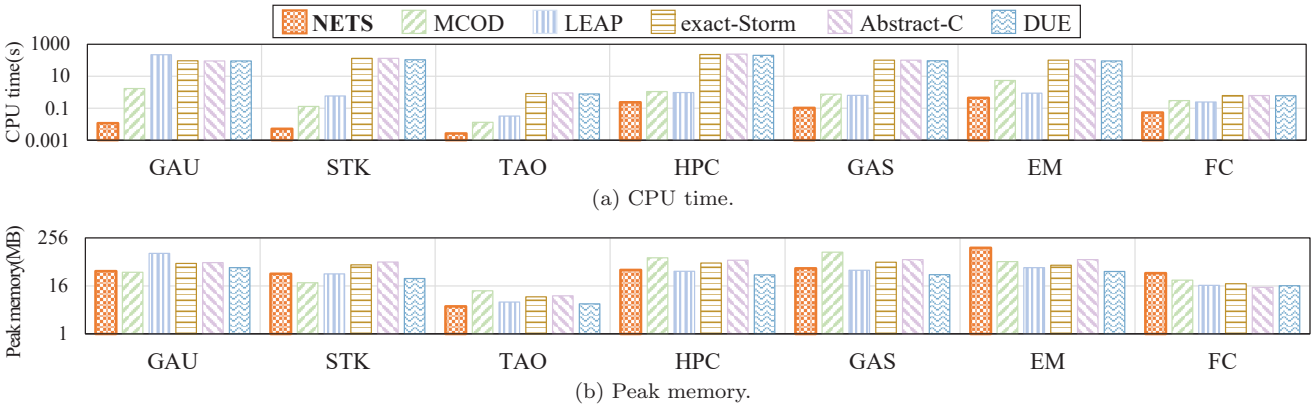
Outlier detection cost via two-level dimensional filtering: We now perform a ballpark analysis on the cost of identifying outliers. The cost model is to estimate the number of D-cells or data points accessed in the middle of outlier detection, because it is heavily related to the performance according to our experience. Also, we assume that data points are uniformly distributed in a domain space, because it is impractical to know data distribution.

Let us consider a set X of data points, and let $\mathbb{C}_{sub}$ be the set of D_{sub}-cells that contain X and $\mathbb{C}_{full}$ be the set of D_{full}-cells that contain X. Then, the cost for detecting outliers via filtering in a sub-dimensional space D_{sub} is modeled as

$$Cost = \overbrace{|\mathbb{C}_{sub}|}^{\text{Step 1}} + \overbrace{\left(\frac{\sum_{c\in\mathbb{C}_{sub}}|N(c)|}{|\mathbb{C}_{sub}|} \times \frac{|\mathbb{C}_{full}|}{|\mathbb{C}_{sub}|}\right)}^{\text{Step 2}} + \underbrace{\left(\frac{\sum_{c\in\mathbb{C}_{sub}}|N(c)|}{|\mathbb{C}_{sub}|} \times \frac{|X|}{|\mathbb{C}_{sub}|}\right)}_{\text{Step 3}}. \quad (1)$$

Following the order of presentation in Section 4, the first term estimates the number of D_{sub}-cells accessed in Step 1, the second term estimates the number of D_{full}-cells accessed in Step 2, and the third term estimates the number of data points accessed in Step 3.

Selection of optimal sub-dimensions: Algorithm 5 outlines the procedure for selecting optimal sub-dimensions. It first takes a set of sample data points from the input stream (Line 1), and then calculates the VMR of each dimension and sorts the dimensions in an increasing order of VMR (Line 2). The estimated cost of outlier detection in the full-dimensional space is computed upfront by Eq. (1) (Line 3). Then, it performs forward selection of dimensions until the estimated cost, Eq. (1), is not decreased (Lines 4–7). Note that the algorithm does not necessarily have to inspect all dimensions because the cost starts increasing when the number of sub-dimensions exceeds a certain balancing point, as a result of the trade-off shown in Figure 6. Once optimal sub-dimensions are determined, the cost estimated for the sub-dimensions is compared with the cost estimated for the full-dimensions, and the set of the dimensions with the smaller cost is returned (Lines 8–12).



(a) CPU time.

(b) Peak memory.

Figure 8: Overall performance for all data sets with the default parameter values. NETS outperforms MCOD by 5 times (TAO) to 150 times (GAU) while consuming only comparable memory space.

6. EXPERIMENTS

Thorough experiments have been conducted to evaluate the performance of NETS. As a result, the following advantages have been confirmed.

- NETS is much faster than state-of-the-art algorithms and requires only comparable memory space (Section 6.2).
- NETS is robust to the variation of performance parameters such as the window size (θ_W), the slide size (θ_S), the distance threshold (θ_R), and the neighbor count threshold (θ_K) (Section 6.3).
- NETS benefits significantly from the set-based update and the two-level dimensional filtering for its high performance (Section 6.4).

6.1 Experiment Setup

Data sets: We used the six real-world data sets in Table 1 and a synthetic data set. Dimensionality of the data sets ranges from 1 to 55. GAU, STK, and TAO are low-dimensional (1 to 3), where GAU [22] is generated by a Gaussian mixture model with three distributions, STK [19] contains stock trading records, and TAO [18] contains oceanographic data provided by the Tropical Atmosphere Ocean project. HPC and GAS are mid-dimensional (7 to 10), where HPC [9] contains electric power consumption data and GAS [9] contains household gas sensor data. EM [9] and FC [9] are high-dimensional (16 to 55), where EM contains chemical sensor data and FC contains forest cover type data. Note that the relatively low concentration ratios of these two data sets (see Table 1) drives NETS to kick off the sub-dimensional filtering option. All data sets except GAS are also used in other researches [2, 4, 14, 22].

Parameters: The main control parameters in DODDS are θ_W (window size), θ_S (slide size), θ_R (distance threshold), and θ_K (number of neighbors threshold). As suggested in the survey by Tran et al. [22], the default values of the parameters for each data set is set to make the ratio of outliers to be approximately 1%. Table 4 summarizes the data sets and corresponding default parameter values.

Algorithms: We chose five algorithms, MCOD [14], LEAP [4], exact-Storm [2], Abstract-C [23], and DUE [14], for comparison with NETS. The five algorithms have been re-implemented in JAVA by Tran et al. [22], and the source codes are available at http://infolab.usc.edu/Luan/Outlier. NETS was also implemented in JAVA, and the source code is available at https://github.com/kaist-dmlab/NETS.

Table 4: Data sets and default parameter values.

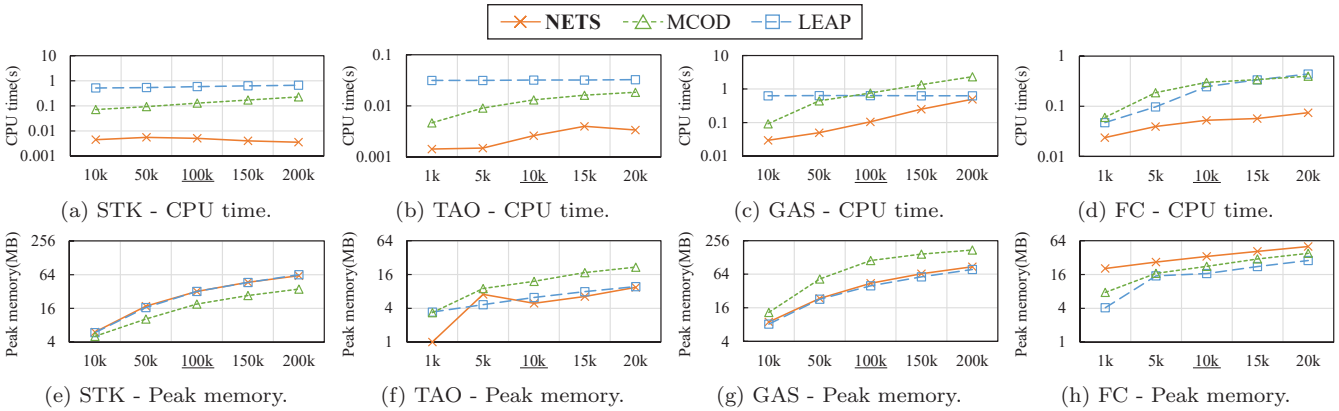
Data set	Dim	Size	θ_W	θ_S	θ_R	θ_K
GAU [22]	1	1.0M	100,000	5,000	0.028	50
STK [19]	1	1.1M	100,000	5,000	0.45	50
TAO [18]	3	0.6M	10,000	500	1.9	50
HPC [9]	7	1.0M	100,000	5,000	6.5	50
GAS [9]	10	0.9M	100,000	5,000	2.75	50
EM [9]	16	1.0M	100,000	5,000	115	50
FC [9]	55	0.6M	10,000	500	525	50

Performance metrics: In an outlier detection application, it is critical to reducing latency in updating outlier information. Additionally, memory consumption should be small enough to work reliably in a commodity machine. We measured the average CPU time and the maximum memory consumed (or peak memory) to update outlier information in every window. We used JAVA ThreadMXBean interface to measure CPU time and a separate thread to measure peak memory, which is the same way as used in the survey [22].

Computing platform: We conducted experiments on an Amazon AWS c5d.xlarge instance with four vCPUs (3GHz), 8GB of RAM, and 100GB of SSD. Ubuntu 18.04.1 LTS and JDK 1.8.0_191 are installed in the instance.

6.2 Highlight of the Results

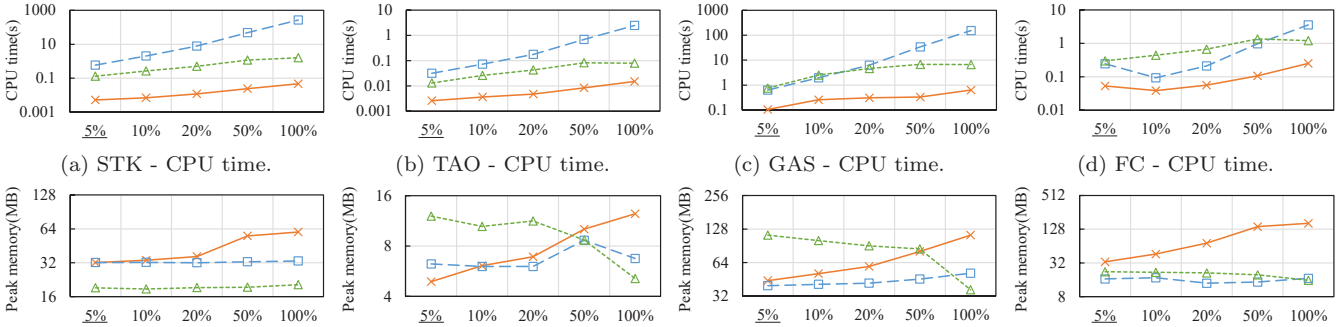
We compare the overall performance of the six algorithms for all data sets with the parameter values set to the defaults shown in Table 4. Figure 8 shows the CPU time and peak memory of the six algorithms. Note the logarithmic scale of the performance numbers. Evidently, NETS was by far the fastest for all data sets, outperforming MCOD by 10 times, LEAP by 24 times, exact-Storm by 4,727 times, Abstract-C by 4,680 times, and DUE by 3,826 times when averaged over all real-world data sets. The three existing algorithms, exact-Storm, Abstract-C, and DUE, were shown to be inferior to the top two state-of-the-art algorithms, MCOD and LEAP. Thus, the subsequent evaluation focuses on comparison with only MCOD and LEAP. Especially for GAU, where both MCOD and LEAP took longer than 1.0s to detect outliers, NETS took only 0.01s, more than 100 times faster. Even for TAO, where both MCOD and LEAP took only 10ms to 40ms, the least among all data sets, NETS took only 2.6ms, still 4 to 15 times faster. Moreover, the peak memory of NETS was similar to that of MCOD and LEAP, specifically the smallest in TAO and no more than 1.4 times larger when averaged over the other data sets. This remark-



(a) STK - CPU time. (b) TAO - CPU time. (c) GAS - CPU time. (d) FC - CPU time.

(e) STK - Peak memory. (f) TAO - Peak memory. (g) GAS - Peak memory. (h) FC - Peak memory.

Figure 9: Varying window size θ_W.



(a) STK - CPU time. (b) TAO - CPU time. (c) GAS - CPU time. (d) FC - CPU time.

(e) STK - Peak memory. (f) TAO - Peak memory. (g) GAS - Peak memory. (h) FC - Peak memory.

Figure 10: Varying slide size θ_S (percentage of the default θ_W value).

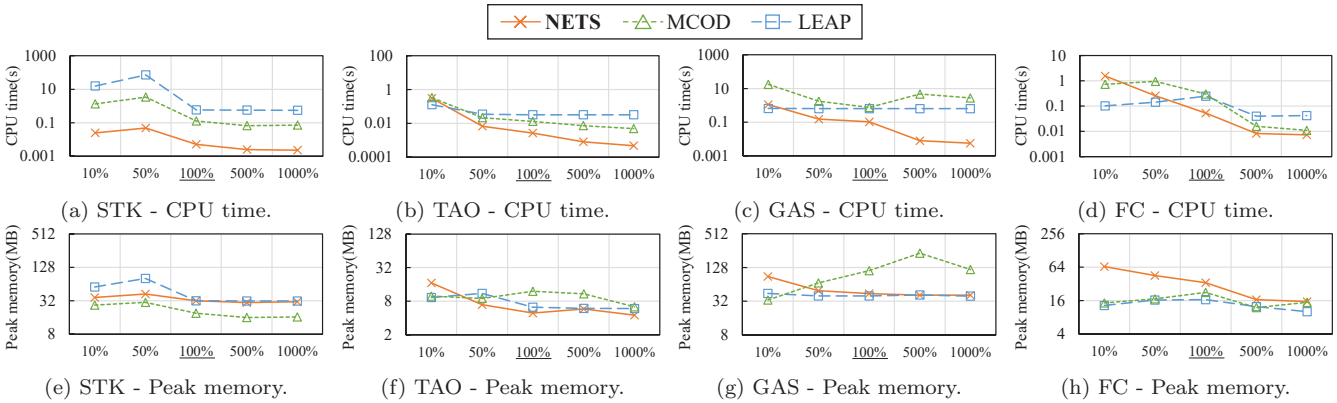
able performance of NETS demonstrates the merits of set-based updates and net effect and additionally the two-level dimensional filtering for EM and FC.

6.3 Effects of Parameters on Performance

We verify the robustness of performance when the parameter values are varying attuned to individual data sets. We present the results for the selected data sets, STK, TAO, GAS, and FC, owing to the lack of space. The results from the other data sets showed similar patterns. In all figures of this section, the default parameter value is underlined.

Varying the window size θ_W (see Figure 9): θ_W is the number of data points in a window and indicates roughly the amount of workload on the algorithms. In this experiment, θ_W was varied from 10K to 200K for STK and GAS and from 1K to 20K for TAO and FC. While CPU time increased with θ_W for all algorithms most of the time, the increase in CPU time for NETS is primarily due to an increase in the number of data points in a D-cell, which results in an increase in the number of data points in non-determined cells. Regardless, NETS was definitely faster than MCOD and LEAP by several orders of magnitude in the entire range of θ_W. This demonstrates that the performance advantage of net effect accompanied by set-based update holds up consistently regardless of varying θ_W. Interestingly, CPU time of NETS for STK (Figure 9a) decreased as θ_W increased. Given that STK has only one dimension, we believe that it happened because an increase in the number of data points in a window directly causes an increase in the number of inlier cells and a decrease in the number of non-determined cells, thus increasing the benefit of early detection. Like CPU time, peak memory also increases with θ_W for all algorithms, which is obvious as the number of data points in a window increases. It is impressive that the peak memory of NETS is no larger than and even smaller than MCOD and LEAP for most data sets. FC (Figure 9h) is an exception, where NETS has higher peak memory than both MCOD and LEAP, although still less than 64MB at the maximum. The reason lies in FC being high-dimensional and therefore needing to keep D_{sub}-cells in memory for sub-dimensional filtering. Note that, in return, NETS runs 4.2 times faster than LEAP and 4.9 times faster than MCOD on average over all θ_W values.

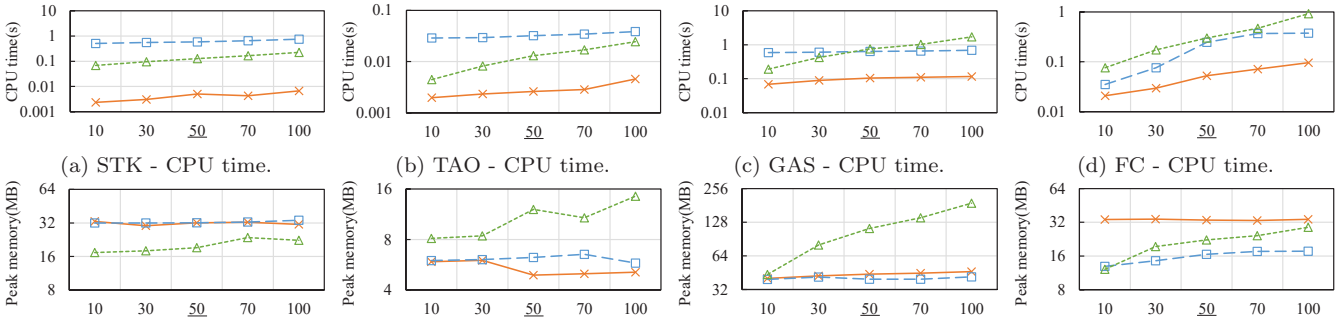
Varying the slide size θ_S (see Figure 10): θ_S determines the number of expired and new data points in each update. In this experiment, θ_S was varied from 5% to 100% of the default θ_W value for each data set. The CPU time of all algorithms clearly increased with θ_S. It happened because, when θ_S is larger, a larger portion of data points in a window is affected by expired or new neighbors and, therefore, the algorithms spend more time updating neighbors and identifying outliers. Here again, NETS achieved by far the smallest CPU time among all three algorithms for all data sets in the entire range of parameter values. It demonstrates the benefit of considering the net effect for updates, thus removing redundant updates. Additionally, NETS showed the slowest growth rate of CPU time. For instance, in the case of GAS (Figure 10c), the CPU time increased 6.1 times (from 0.11s to 0.64s) in NETS whereas 8.8 times (from 0.76s to 6.7s) in MCOD and 243 times (from 0.64s to 154s) in LEAP. The peak memory of NETS always increased with θ_S, because a larger θ_S makes NETS use more memory to derive the net effect between the expired slide and the new slide. One plausible observation is that for most data sets NETS used more memory than MCOD and LEAP when θ_S was more than 50% (of the window size).



(a) STK - CPU time. (b) TAO - CPU time. (c) GAS - CPU time. (d) FC - CPU time.

(e) STK - Peak memory. (f) TAO - Peak memory. (g) GAS - Peak memory. (h) FC - Peak memory.

Figure 11: Varying distance threshold θ_R (percentage of the default θ_R value).



(a) STK - CPU time. (b) TAO - CPU time. (c) GAS - CPU time. (d) FC - CPU time.

(e) STK - Peak memory. (f) TAO - Peak memory. (g) GAS - Peak memory. (h) FC - Peak memory.

Figure 12: Varying number of neighbors threshold θ_K.

Varying the distance threshold θ_R (see Figure 11): θ_R determines the area of neighborhood so that a higher θ_R includes more data points as neighbors. In this experiment, θ_R was varied from 10% to 1000% of the default θ_R value for each data set. Both CPU time and peak memory decreased in general for all three algorithms as θ_R increased. Since a larger θ_R makes data points keep more neighbors in a window, inliers are less likely to become outliers even if they lose some of their neighbors. So, with fewer outliers to be detected, the CPU time naturally decreases. The decrease in peak memory is attributed to the decrease in the number of D-cells to manage in NETS and in the number of potential outliers (hence the size of the trigger list and the event queue) in MCOD and LEAP. NETS achieved much smaller CPU time than MCOD and LEAP in almost the entire range of parameter values (e.g., $> 10\%$). This performance advantage, however, was diminished when the value of θ_R was very small (i.e., 10% of the default value) for most data sets except STK, because a smaller θ_R leads to a smaller D-cell size and hence a smaller number of data points in a D-cell. Therefore, the benefit of set-based update and net effect are diminished when θ_R is very small. In the opposite case, however, when θ_R was very large (i.e., 1000% of the default value), NETS benefited from a very high concentration ratio. As a result, in the case of TAO (Figures 11b and 11f) for instance, NETS outperformed LEAP and MCOD in CPU time up to 68 times and 10 times, respectively, and NETS consumed only 4.5MB peak memory, which was only 76% of 5.9MB in LEAP and 71% of 6.4MB in MCOD.

Varying the number of neighbors threshold θ_K (see Figure 12): Since θ_K is the minimum number of neighbors required for a data point to be an inlier, a higher θ_K makes more data points outliers. In this experiment, θ_K was varied from 10 to 100 for all data sets. Here again, NETS was definitely the fastest algorithm in the entire range of parameter values for all data sets, while consuming lower or similar memory compared with MCOD and LEAP except in FC. The CPU time of NETS increases with θ_K because a higher θ_K makes more D-cells have fewer than θ_K data points, which results in fewer inlier cells and more non-determined cells. Interestingly, peak memory of NETS was almost constant for varying θ_K. This is because θ_K affects neither the number of D-cells updated in a window (as θ_W or θ_S does) nor the number of D-cells (as θ_R does). Thus, NETS uses constant memory space to manage a window for varying θ_K.

6.4 Efficacy of Set-Based Update and Two-Level Dimensional Filtering

Set-based update: This technique enables NETS to efficiently filter out inlier cells and outlier cells by only updating the net effect, the net-change in D-cell cardinality. Moreover, the net effect in turn enables NETS to identify inlier data points from non-determined cells. Consequently, only a small portion of data points are required to find additional neighbors. Table 5 shows for each data set the ratio of each of the three types (i.e., inlier, outlier, and non-determined) of data points, averaged over all windows based on the default parameter values.

Note that, in the case of low- to mid-dimensional data sets (i.e., GAU, STK, TAO, HPC, and GAS), more than 98% of data points were identified early as inliers or outliers by only updating the net effect. Even for the high-dimensional data sets (i.e., EM and FC), the ratio was around 90%. By exploiting the net effect, NETS needed to inspect only a few data points further to find additional neighbors in order to identify outliers.

Table 5: Average ratio(%) of each type of data points in a window. X_{in}, X_{out}, and X_{non} indicate inlier, outlier, and non-determined data points, respectively.

Type	GAU	STK	TAO	HPC	GAS	EM	FC
X_{in}	98.7	99.5	98.7	98.4	98.7	94.1	89.2
X_{out}	0.80	0.37	0.40	0.20	0.80	0.30	0.40
X_{non}	**0.50**	**0.13**	**0.90**	**1.40**	**0.50**	**5.60**	**10.4**

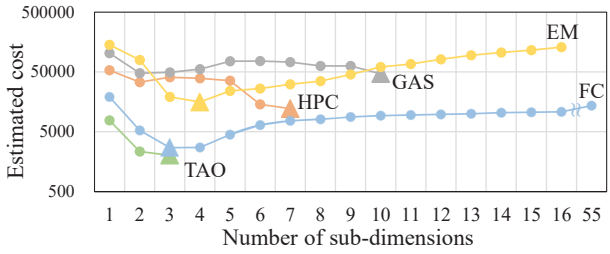
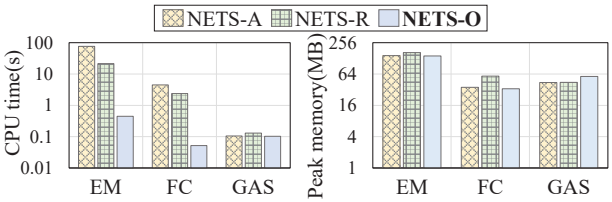


Figure 13: Estimated cost against the number of sub-dimensions. (The lowest point for each data set is marked with a triangle.)



(a) CPU time. (b) Peak memory.

Figure 14: The effect of optimal sub-dimensions.

Two-level dimensional filtering: We did two kinds of analysis. First, to show how optimal sub-dimensions are selected (see Section 5.2), we plotted the cost calculated using Eq. (1) against the number of sub-dimensions. Figure 13 shows the results for all data sets that have two or more dimensions. For EM and FC, the optimal sub-dimensions that give the lowest cost were, respectively, the top three and top four prioritized dimensions. Attributed to the trade-off discussed in Section 5.1, the cost decreased initially as the number of sub-dimensions increased and then increased after the optimal point. For the other data sets, the full dimensions resulted in the lowest cost because they already had a high concentration ratio in full dimensions.

Second, to examine the effectiveness of sub-dimensional filtering in the selected optimal sub-dimensions, we designed three variations of NETS: NETS-A, NETS-R, and NETS-O. NETS-A is the baseline algorithm that does not use sub-dimensional filtering at all. NETS-R and NETS-O perform sub-dimensional filtering in *random* sub-dimensions and *optimal* sub-dimensions, respectively. For NETS-R, we report the average from ten repeated executions. Figure 14 shows the performances of the three variations of NETS for GAS, EM, and FC. Adopting sub-dimensional filtering (NETS-R and NETS-O) was especially useful when the original concentration ratio was low, as in EM and FC whose concentration ratios are 0.42 and 0.44. The CPU time in EM and FC was significantly reduced in exchange for a little more or similar amount of memory space. However, there was almost no improvement for GAS since it already had a high concentration ratio (i.e., 0.88) even in full dimensions. Furthermore, NETS-O outperformed NETS-R for both EM and FC,

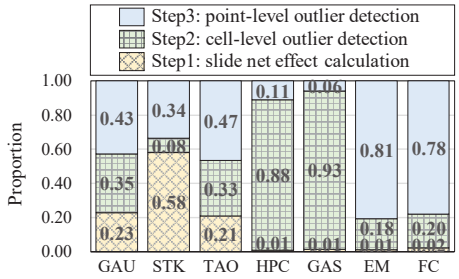


Figure 15: The breakdown of NETS CPU time.

which indicates that the optimally selected sub-dimensions balanced out the trade-off better than the randomly selected sub-dimensions.

The breakdown of NETS: Figure 15 shows the breakdown of the CPU time of NETS into the three steps discussed in Section 4. It shows an interesting contrast in the proportion of the steps depending on the dimensionality of a data set. For the low-dimensional ones (i.e., GAU, STK, and TAO), the three steps seem to be relatively balanced. (Step 2 of STK seems a bit exceptional, taking a significantly lower portion than the other steps, which indicates a smaller number of D-cells in the window.) For the mid-dimensional ones (i.e., HPC and GAS), Step 2 has a dominant proportion, indicating that the cell-level outlier detection is the biggest source of the performance gain achieved by NETS. For the high-dimensional ones (i.e., EM and FC), Step 3 has a dominant proportion, which indicates that the high dimensionality still has a lingering effect toward deferring outlier detection to the costly point-level detection step. (Note that, still, only approximately 10% of all data points were inspected in Step 3, as shown in Table 5.)

7. CONCLUSION

In this paper, we proposed NETS, a very fast novel distance-based outlier detection algorithm for data streams. By verifying that the effects of expired and new data points can be aggregated or canceled-out in data streams, we proposed the set-based update to exploit such net effect. NETS calculates the net effect of changed data points by grouping them as cells. Most data points can be quickly identified as outliers or inliers by only using the net effect at the cell level. To efficiently update outliers even from a sparsely distributed data stream, two-level dimensional filtering was adopted to exploit a higher concentration ratio inside systematically chosen sub-dimensions. NETS outperformed state-of-the-art algorithms by several orders of magnitude in CPU time for most real-world data sets while consuming only comparable memory space. We believe that the proposed approach makes it possible to detect outliers much faster in real-time and also opens a new research direction for outlier detection from data streams.

8. ACKNOWLEDGMENTS

This work was partly supported by the MOLIT (The Ministry of Land, Infrastructure and Transport), Korea, under the national spatial information research program supervised by the KAIA (Korea Agency for Infrastructure Technology Advancement) (19NSIP-B081011-06) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (No. 2017R1E1A1A01075927).

9. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[2] F. Angiulli and F. Fassetti. Detecting distance-based outliers in streams of data. In *Proceedings of 16th ACM Conference on Conference on Information and Knowledge Management*, pages 811–820, 2007.

[3] L. Cao, Q. Wang, and E. A. Rundensteiner. Interactive outlier exploration in big data streams. *PVLDB*, 7(13):1621–1624, 2014.

[4] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner. Scalable distance-based outlier detection over high-volume data streams. In *Proceedings of 2014 IEEE 30th International Conference on Data Engineering*, pages 76–87, 2014.

[5] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:72, 2009.

[6] C.-H. Cheng, A. W. Fu, and Y. Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 84–93, 1999.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Data Bases*, pages 426–435, 1997.

[8] D. Cox and P. Lewis. The statistical analysis of series of events. *John Wiley and Sons*, 1966.

[9] D. Dua and C. Graff. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2019.

[10] D. Georgiadis, M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. Continuous outlier detection in data streams: An extensible framework and state-of-the-art algorithms. In *Proceedings of 2013 ACM SIGMOD International Conference on Management of Data*, pages 1061–1064, 2013.

[11] J. Han, J. Pei, and M. Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2011.

[12] P. Kakria, N. K. Tripathi, and P. Kitipawang. A real-time health monitoring system for remote cardiac patients using smartphone and wearable sensors. *International Journal of Telemedicine and Applications*, 2015:1–11, 2015.

[13] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 392–403, 1998.

[14] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. Continuous monitoring of distance-based outliers over data streams. In *Proceedings of 2011 IEEE 27th International Conference on Data Engineering*, pages 135–146, 2011.

[15] J.-G. Lee and M. Kang. Geospatial big data: Challenges and opportunities. *Big Data Research*, 2(2):74–81, 2015.

[16] Y. Lin, B. S. Lee, and D. Lustgarten. Continuous detection of abnormal heartbeats from ECG using online outlier detection. In *Proceedings of 2018 International Symposium on Information Management and Big Data*, pages 349–366, 2018.

[17] N. G. Mankiw. *Principles of Economics*. Cengage Learning, 2014.

[18] NOAA. Tropical atmosphere ocean project. https://www.pmel.noaa.gov, 2019. Accessed: 2019-03-01.

[19] U. of Pennsylvania. Wharton research data services. https://wrds-web.wharton.upenn.edu/wrds/, 2019. Accessed: 2019-03-01.

[20] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations Newsletter*, (1):90–105, 2004.

[21] S. Sadik, L. Gruenwald, and E. Leal. Wadjet: Finding outliers in multiple multi-dimensional heterogeneous data streams. In *Proceedings of 2018 IEEE 34th International Conference on Data Engineering*, pages 1232–1235, 2018.

[22] L. Tran, L. Fan, and C. Shahabi. Distance-based outlier detection in data streams. *PVLDB*, 9(12):1089–1100, 2016.

[23] D. Yang, E. A. Rundensteiner, and M. O. Ward. Neighbor-based pattern detection for windows over streaming data. In *Proceedings of 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 529–540, 2009.

[24] I. Yi, J.-G. Lee, and K.-Y. Whang. APAM: Adaptive eager-lazy hybrid evaluation of event patterns for low latency. In *Proceedings of 25th ACM Conference on Conference on Information and Knowledge Management*, pages 2275–2280, 2016.