Multiple Dynamic Outlier-Detection from a Data Stream by Exploiting Duality of Data and Queries

Susik Yoon
KAIST
susikyoon@kaist.ac.kr

Yooju Shin
KAIST
yooju24@kaist.ac.kr

Jae-Gil Lee*
KAIST
jaegil@kaist.ac.kr

Byung Suk Lee
University of Vermont
bslee@uvm.edu

ABSTRACT

Real-time outlier detection from a data stream has become increasingly important in the current hyperconnected world. This paper focuses on an important yet unaddressed challenge in continuous outlier detection—the *multiplicity* and *dynamicity* of queries. This challenge arises from various contexts of outliers evolving over time, but the state-of-the-art algorithms cannot handle the challenge effectively, as they can only process a *fixed set* of outlier detection queries for each data point *separately*. In this paper, we propose a novel algorithm, abbreviated as **MDUAL**, based on a new idea called *duality-based unified processing*. The underlying rationale is to exploit the *duality of data and queries* so that a group of similar data points are processed together by a group of similar queries incrementally. Two main techniques embodying the idea, *data-query grouping* and *prioritized group processing*, are employed. Comprehensive experiments showed that MDUAL runs 216 to 221 times faster while consuming 11 to 13 times less memory than the state-of-the-art algorithms through its efficient and effective handling of the multiplicity–dynamicity challenge.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; • **Information systems** → **Data stream mining**.

KEYWORDS

Outlier detection, anomaly detection, data stream

ACM Reference Format:
Susik Yoon, Yooju Shin, Jae-Gil Lee, and Byung Suk Lee. 2021. Multiple Dynamic Outlier-Detection from a Data Stream by Exploiting Duality of Data and Queries. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3452810

1 INTRODUCTION

1.1 Background and Motivation

With the advent of hyperconnected society where people, devices, and facilities are interconnected through seamless network communications, real-time outlier detection from a data stream is gaining increasing attention [11, 16, 32, 36]. Outliers are data instances with unusual characteristics that cannot be found in the majority of other data instances [2], and detecting them in real time is crucial in many applications, such as detecting fraudulent activities in financial transactions [1] or monitoring for abnormal beats in a patient's ECG [25].

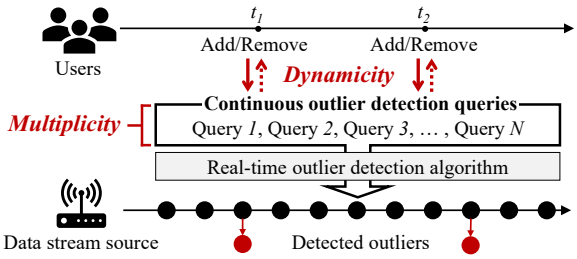


Figure 1: Real-time outlier detection with multiple and dynamic outlier detection queries.

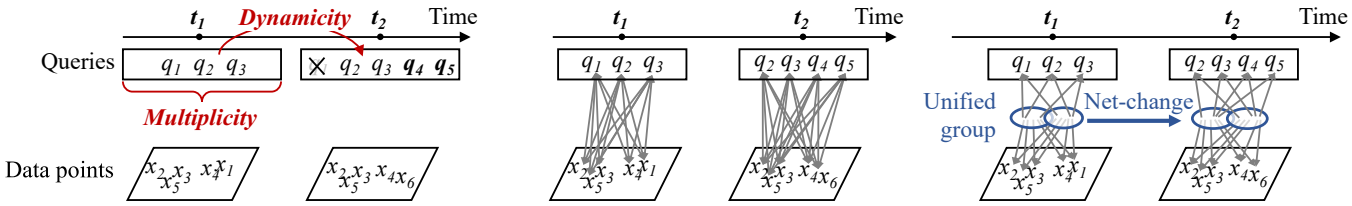
Real-time outlier detection from a data stream typically involves continuously running queries that filter out data points that satisfy outlier conditions defined on parameters, and in this regard the query processing must be efficient to support real-time detection and effective to achieve accurate detection. With this general challenge as the backdrop, this paper focuses on a newly identified, yet understudied, challenge of queries originated from the complex nature of data streams today, namely the *multiplicity* and *dynamicity* of continuous outlier detection queries, as illustrated in Figure 1. While running multiple and dynamic continuous queries has been well studied in other systems such as a complex event processing (CEP) system [24, 33, 43] and a publish-subscribe (Pub-Sub) system [10, 27], it has not been studied much for an outlier detection system, which often examines complex combinations of various conditions like similarity to other data points and temporal span of interest, and thus a new study is needed.

- *Multiplicity*: Due to various analytic purposes of real-time outlier detection by different users in different circumstances, an outlier detection algorithm may need to handle multiple queries concurrently. There are plenty of such situations including, but not limited to, financial fraud detection [6, 35], enterprise security surveillance [14, 37], healthcare monitoring [18], or online outlier exploration [7, 8]. Specifically, in financial fraud detection applications, analysts look for unusual transactions that are significantly different from a majority of recent transactions, wherein individual analysts may well issue queries with varying parameter values to capture differing degrees of "significance," "majority," and "recency" in determining the unusualness of transactions [6, 35]. The number of concurrent queries may vary widely, but commonly reaches 100s to 1000s, as exemplified in the "big query" environment [15, 34, 44].

*Jae-Gil Lee is the corresponding author.

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452810



(a) Data points and continuous outlier detection queries. (b) Separate processing. (c) Duality-based unified processing.

Figure 2: Two approaches for processing multiple and dynamic outlier detection queries.

Table 1: Comparison of existing algorithms' ability to handle the multiplicity and dynamicity challenges.

	MCOD [21]	LEAP [9]	NETS [40]	SOP [6]	pMCSKY [35]	**Ours**
Multiplicity	×	×	×	O	O	**O**
Dynamicity	×	×	×	×	×	**O**

- *Dynamicity*: Due to evolving circumstances characterizing outliers over time, the composition of outlier detection queries can vary dynamically. For instance, in credit card fraud detection where there are massive transactions changing over time, there can also be a large number of outlier detection queries changing over time to detect various abnormal transactions. The abnormality could be, for example, sudden above-market payments, indicating hoarding activities during the recent pandemic lockdown; recurring micro-payments for items typically paid at once, indicating a possible tax evasion attempt after a revised tax law; and suspicious changes in credit card usage patterns, indicating possible new fraud techniques [1, 38]. Detecting these abnormalities changing over time would necessitate outlier detection queries dynamically specifying different conditions for capturing different payment patterns added or dropped over time. Thus, an outlier detection algorithm is required to handle such a non-stationary set of outlier detection queries adaptively.

The challenge of processing multiple and dynamic outlier detection queries continuously in real time is beyond the limit of the existing outlier detection algorithms [6, 9, 21, 22, 35, 40]. As summarized in Table 1, some [9, 21, 40] are designed for only a single (not multiple) outlier detection query, and the others [6, 22, 35] consider only fixed (not dynamic) outlier detection queries. Figure 2a shows example data points and multiple outlier detection queries changing dynamically between two consecutive timestamps (t_1 and t_2). In the existing algorithms, each outlier detection query operates on each data point (to check if it is an outlier) at each timestamp; thus, in this example, 15 (= 3×5) operations are needed at time t_1, and 20 (= 4×5) operations at time t_2 (see Figure 2b). While SOP [6] and pMCSKY [35] tried to reduce the number of operations with shared data structure for multiple operations or early pruning of non-outliers, both algorithms still incur significantly redundant query operations.

Therefore, the goal of this paper is to propose a novel algorithm that can handle both the multiplicity and dynamicity efficiently (for real time) and effectively (for accuracy). Obviously, the key to achieving this goal lies in exploiting the ***data similarity*** and ***query similarity*** towards removing the redundant computations as much as possible. That is, similar data points should be processed together, and similar queries should be, too. These similarities are frequently observed in real-world environments. A real-world data stream shows high data similarity since the data distribution is highly skewed [23, 40]. At the same time, outlier detection queries issued by a cohort of users show high similarity since users in the same cohort tend to show a herd behavior [31].

1.2 Main Idea: Duality-based Unified Processing

Given a set of data-query pairs to be processed, our algorithm exploits the dual relationship between data and queries in grouping the data-query pairs. That is, the pairs of *similar data points* can be grouped and, likewise, the pairs of *similar queries* can be grouped. Ultimately, these two similarities are unified so that the data-query pairs are grouped together based on both similarities, thus enabling *duality-based unified processing* and minimizing redundant computations. In Figure 2c, for example, all data-query pairs are divided into two *unified groups* to be processed collectively, thereby sharing the results of overlapping operations within the group. Furthermore, only the net changes of queries and data points in each unified group are updated between consecutive timestamps to save the repetitive operations. Thus, the former (the grouped processing) is effective for handling the multiplicity while the latter (the net-change-based update) is for the dynamicity.

To demonstrate the idea of duality-based unified processing in this paper, we adopted distance-based [2, 20] outlier detection, one of the representative outlier detection mechanisms. A data point x in a stream becomes an outlier if it has fewer than a certain number of data points within a certain distance from x in the most recent temporal context [36]. This mechanism has been widely adopted in many real-time outlier detection applications because it is intuitive and shows robust performance across various domains [6, 9, 21, 22, 35, 40]. Applying the main idea to the distance-based outlier detection by multiple dynamic outlier detection queries gives the algorithm **MDUAL** (Multiple Dynamic outlier-detection from a data stream by exploiting DUALity of data and queries) discussed in this paper. There are two key techniques instrumental in this algorithm: *data-query grouping* and *prioritized group processing*.

- *Data-query grouping*: Each outlier detection query in the query set (or space) is projected onto each data point in the data space, and then the projected queries are divided into groups. Then, the projected queries in each group are processed collectively under a single operation that subsumes all overlapping operations inside the group so that its result is sufficient to derive exact answers for all projected queries in the same group.

Table 2: Notations used in this paper.

Notation	Description
x	a data point
W	a window
c	a cell
$card(c)$	the number of data points in a cell c
q	a continuous outlier detection query
$q.r$	the distance threshold of a query q
$q.k$	the neighbor count threshold of a query q
$q.w$	the window size of a query q
$q.s$	the slide size of a query q
Q	an outlier detection query set
q^{cbr}	a constrained Boolean range query (CBRQ)
UG	a unified group
UGM	a unified group mapping

- *Prioritized group processing*: The processing of projected queries inside a group is further accelerated by exploiting dependencies between the projected queries. That is, when the results of certain projected queries in a group imply the results of other projected queries in the same group, MDUAL prioritizes the projected queries in the group and answers the one with the highest priority first and propagates its result to the others.

1.3 Highlights

Main contributions in this paper are summarized as follows.

- To the best of our knowledge, this is the first work to exploit the duality of data and queries for processing multiple dynamic outlier detection queries from a data stream.
- We propose the novel algorithm MDUAL to implement duality-based unified processing using two techniques, data-query grouping and prioritized group processing, with concrete theoretical supports. Its source code is available in GitHub [42].
- We empirically show that MDUAL processes multiple dynamic outlier detection queries 216 to 221 times faster while consuming 11 to 13 times less memory than the two state-of-the-art algorithms, SOP [6] and pMCSKY [35], in comprehensive experiments with six real data sets.

In the rest of the paper, Section 2 formalizes the problem, Section 3 introduces the main techniques of MDUAL, Section 4 discusses the proposed MDUAL algorithm, Section 5 presents the experiments, Section 6 reviews related work, and Section 7 concludes the paper.

2 PROBLEM SETTING

A *data stream* is an unbounded sequence of data points, ..., x_{t-1}, x_t, x_{t+1}, ..., arriving continuously at each timestamp. We adopt the concept of a *sliding window* to work with a data stream, where a *window* refers to the set of the most recent data points and a *slide* refers to a set of data points added to or removed from a window as it slides. The window size and the slide size are commonly defined as the number of data points, while alternatively can be defined as the duration [36].

A *distance-based outlier* [2, 20] is a data point having only few other data points within a certain distance. Specifically, given a set of data points, a data point is a distance-based outlier (or simply *outlier*) if it has less than k other data points, called *neighbors*, within the distance r, and otherwise a distance-based inlier (or simply *inlier*). The Euclidean distance, denoted as $dist(x_1, x_2)$, is commonly used to calculate the distance between two data points x_1 and x_2.

Now, let us first formalize a simple outlier detection problem from a data stream with a single stationary continuous outlier detection query (see Definition 2.1) and then extend it to the problem with multiple dynamic continuous outlier detection queries (see Definition 2.2), which is the subject of this paper. Table 2 summarizes the notations used in this paper.

Definition 2.1. (DODDS[36]) Let q be a continuous outlier detection query (or *outlier query* in short) defined by four conditions $\langle r, k, w, s\rangle$ which mean the distance threshold, the neighbor count threshold, the window size, and the slide size, respectively. Given a data stream and an outlier query q, the problem of *distance-based outlier detection in data streams (DODDS)* is to find the outliers defined by $q.r$ and $q.k$ in every window of size $q.w$ sliding by $q.s$. □

Definition 2.2. (MD-DODDS) Given a data stream and a set Q of outlier queries changing dynamically over time, the problem of *multiple dynamic query processing for distance-based outlier detection in data streams (MD-DODDS)* is to find all outliers detected by all outlier queries in Q. □

3 DUALITY-BASED UNIFIED PROCESSING

3.1 Data-Query Grouping

3.1.1 Integrating Data and Outlier Queries. In MD-DODDS, data points and outlier queries are formulated in two distinct spaces: a d-dimensional *data* space defined by d data attributes and a four-dimensional *query* space defined by four outlier query condition attributes r, k, w, and s. Each outlier query q_i in the query space is projected onto each data point x_j in the data space in the form of a *range query* $q_i^r(x_j)$, which returns all data points within the distance $q_i.r$ from x_j. The range query turns to a Boolean query that indicates the outlierness (true or false) of a given point by a given outlier query (see Definition 3.1).

Definition 3.1. (Constrained Boolean Range Query (CBRQ)) Given a data point x_j and an outlier query q_i, the *constrained Boolean range query (CBRQ)* of q_i at x_j, denoted as $q_i^{cbr}(x_j)$, is a Boolean query indicating whether x_j is an outlier by q_i or not. □

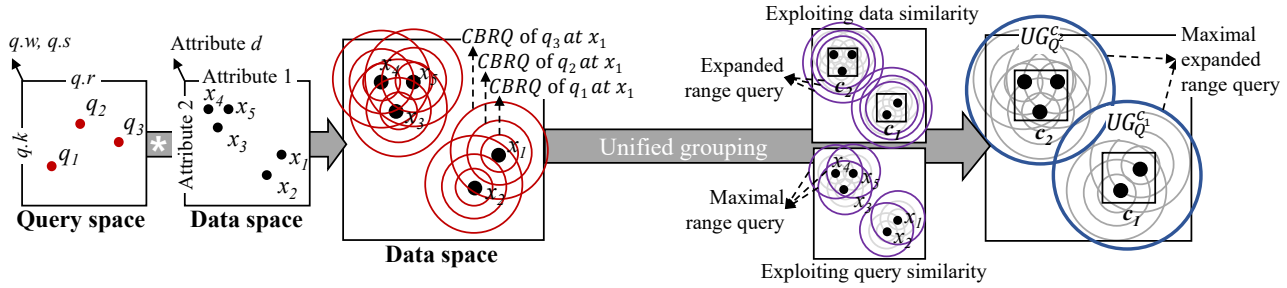
A CBRQ $q_i^{cbr}(x_j)$ is initialized as *null* and calculated as

$$q_i^{cbr}(x_j) = \begin{cases} true & \text{if } |\{x_k \mid x_k \in q_i^r(x_j) \wedge x_k \in \mathbb{W}_{q_i.w}^{q_i.s}\}| \le q_i.k \\ false & \text{otherwise} \end{cases}, \quad (1)$$

where $\mathbb{W}_{q_i.w}^{q_i.s}$ is the current window of size $q_i.w$ sliding by $q_i.s$. That is, $q_i^{cbr}(x_j)$ returns *true* if x_j is an outlier by q_i and *false* otherwise (i.e., if x_j is an inlier by q_i).

Figure 3a illustrates the projection of outlier queries from the query space onto the data space and the resulting CBRQs. There are three outlier queries in the query space and five data points in the data space. By projecting each outlier query against each data point, 15 CBRQs (= 3 outlier queries × 5 data points) are created.

3.1.2 Grouping CBRQs. The CBRQs can be grouped if they have similar conditions. Effective grouping should reduce or remove



(a) CBRQs projected from the query space onto the data space. (b) Unified grouping of CBRQs.

Figure 3: Integration and grouping of data and outlier queries.

repetitive computations without affecting the resulting accuracy of outlier detection. We use the distance threshold ($q.r$) for grouping CBRQs, because it has the highest impact on the run time among the four conditions due to a large number of distance computations, one per data point, in the query processing. Note that, given the duality between data and queries, the grouping can be done by exploiting data and query similarities jointly.

Exploitation of data similarity: The idea is to process similar data points together for each outlier query, by grouping the CBRQs of an outlier query whose range queries have center points that are spatially close. We determine the spatial proximity efficiently via a *cell* obtained by grid partitioning (see Definition 3.2).

Definition 3.2. (Cell) A cell c_i is a hypercube in the data space, specified by the center, $c_i.center$, and the diagonal length, $c_i.size$; the number of data points in a cell is called the cardinality of the cell and denoted as $card(c_i)$; the distance between the centers of two cells, c_1 and c_2, is denoted as $dist(c_1, c_2)$. □

The data points are grouped into a set of non-overlapping and non-empty cells. For each cell c_i, the range queries of the CBRQs of an outlier query whose centers are located within the cell are merged into an *expanded range query*, whose distance threshold is expanded so that the resulting range query subsumes all the range queries merged. For instance, the 15 range queries of the CBRQs in Figure 3a are merged into the six expanded range queries (the upper left in Figure 3b), as the data points are grouped into the two cells by the data similarity.

Exploitation of query similarity: The idea is to process similar outlier queries, of which condition attributes are similar, together for each data point. When we project outlier queries onto a single data point, the resulting range queries naturally have the same centers and entirely overlap one another. Thus, they can be merged into a *maximal range query*, whose distance threshold is set to the largest range so that the maximal range query subsumes all the range queries merged. For instance, the 15 range queries of the CBRQs in Figure 3a are merged into the five maximal range queries (the lower left in Figure 3b) by exploiting the query similarity.

Joint exploitation of data and query similarities: The two ways to exploit the similarities are independent of each other, so synergistic effect can be achieved by jointly grouping CBRQs into *unified groups* by merging the corresponding range queries into a *maximal expanded range query*, whose distance threshold is expanded and maximized so that the resulting range query subsumes all the range queries merged (see Definitions 3.3 and 3.4). The correctness is proven in Theorem 3.5.

Definition 3.3. (Unified Group (UG)) Given a cell c_k and an outlier query set Q, a *unified group* $UG_Q^{c_k}$ is a set of CBRQs of each $q_i \in Q$ at each $x_j \in c_k$. □

Definition 3.4. (Maximal Expanded Range Query) The *maximal expanded range query* of $UG_Q^{c_k}$ is a range query at $c_k.center$ whose distance threshold $r = \max_{q_i \in Q}(q_i.r) + c_k.size$. □

Theorem 3.5. (Correctness of UG) *The result of the maximal expanded range query of $UG_Q^{c_k}$ includes all the results returned by the individual range queries of the CBRQs in $UG_Q^{c_k}$.*

Proof. Let $dist(c_k, x_j)$ be the distance between $c_k.center$ and x_j. Since $c_k.size \geq dist(c_k, x_j)$ for all $x_j \in c_k$ and $\max_{q_i \in Q}(q_i.r) \geq q_i.r$ for all $q_i \in Q$, $\max_{q_i \in Q}(q_i.r) + c_k.size \geq q_i.r + dist(c_k, x_j)$ holds for all $x_j \in c_k$ and all $q_i \in Q$. Thus, the hypersphere occupied by the maximal expanded range query contains all those occupied by individual range queries; so do the data points included in the hypersphere. This naturally leads to the inclusion relationship of query results. □

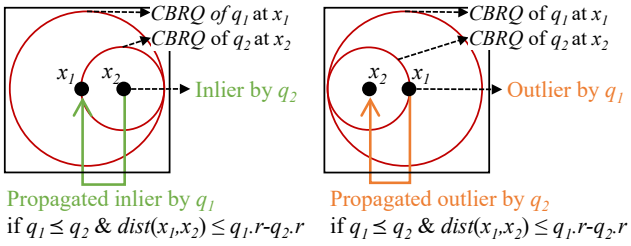
In Figure 3, the 15 CBRQs (Figure 3a) are grouped into the two unified groups, and the corresponding 15 range queries of the CBRQs are merged into the two maximal expanded range queries (the right in Figure 3b).

3.2 Prioritized Group Processing

CBRQ processing: Each CBRQ in a unified group can be processed by pruning the result of the maximal expanded range query of the group with the condition of the CBRQ. This group-wise processing is facilitated by a *unified group mapping* (see Definition 3.6).

Definition 3.6. (Unified Group Mapping (UGM)) A *unified group mapping* $\{\langle c_k : \langle UG_Q^{c_k}, card(c_k), N(c_k)\rangle\rangle\}$ maps a cell c_k to the triplet of the unified group $UG_Q^{c_k}$, the cardinality $card(c_k)$ of the cell, and the set $N(c_k)$ of neighbor cells whose centers fall in the maximal expanded range query of $UG_Q^{c_k}$. □

The triplet in UGM accelerates the calculation of a CBRQ $q_i^{cbr}(x_j)$ of an outlier query q_i at a data point $x_j \in c_k$ by reducing the search space of the neighbors of x_j for processing q_i to the neighbor cells in $N(c_k)$ and bounding the neighbor count to the sum of the cardinalities of the neighbor cells in $N(c_k)$.



(a) Inlier propagation. (b) Outlier propagation.

Figure 4: Examples of propagated inlier/outliers.

CBRQ result propagation: Once the result of a specific CBRQ is known, the results of other CBRQs can be derived from it considering the restrictiveness of outlier query conditions (see Definition 3.7 and Lemma 3.8) and the proximity of data points.

Definition 3.7. (Restrictive relationship of conditions) An outlier query q_1 is more restrictive than another outlier query q_2 in finding outliers (i.e., less likely to return an outlier), denoted as $q_1 \preceq q_2$, if q_1 sets a longer distance threshold for neighbors ($q_1.r \ge q_2.r$), considers a larger window size ($q_1.w \ge q_2.w$), and requires fewer neighbors ($q_1.k \le q_2.k$) than q_2. □

Lemma 3.8. *Let a data point x be an outlier by an outlier query q_1. If $q_1 \preceq q_2$, then x must be an outlier by the outlier query q_2, too. That is, $O_1 \subseteq O_2$, where O_1 and O_2 are respectively the sets of outliers returned by q_1 and q_2 from a set of data points.*

Proof. Let $|q_1^r(x)|$ and $|q_2^r(x)|$ be the neighbor counts of x by q_1 and q_2, respectively. $q_1 \preceq q_2$ indicates that $|q_2^r(x)| \le |q_1^r(x)|$ (from $q_1.r \ge q_2.r$ and $q_1.w \ge q_2.w$) and $q_1.k \le q_2.k$. If x is an outlier by q_1, then $|q_1^r(x)| < q_1.k$ and thus $\boldsymbol{|q_2^r(x)| \le |q_1^r(x)| < q_1.k \le q_2.k}$. Therefore, x must be an outlier by q_2 as well. □

Depending on the result of a CBRQ, either *false* or *true*, the results of other CBRQs can be derived by *inlier propagation* and *outlier propagation* as formulated in Theorems 3.9 and 3.10, respectively, and illustrated in Figure 4.

Theorem 3.9. (inlier propagation property) *Given two data points x_1 and x_2 and two outlier queries q_1 and q_2, $q_2^{cbr}(x_2) = false \Rightarrow q_1^{cbr}(x_1) = false$ holds if $q_1 \preceq q_2$ and $dist(x_1,x_2) \le q_1.r - q_2.r$.*

Proof. As shown in Figure 4a, the number of neighbors of x_1 by q_1 is no less than that of x_2 by q_2 if $dist(x_1,x_2) \le q_1.r - q_2.r$. Thus, Lemma 3.8 can be extended to the case where q_1 and q_2 are respectively applied to x_1 and x_2. Hence, if x_2 is not an outlier by q_2, x_1 cannot be an outlier by q_1, either (by the contrapositive of Lemma 3.8). □

Theorem 3.10. (Outlier propagation property) *Given two data points x_1 and x_2 and two outlier queries q_1 and q_2, $q_1^{cbr}(x_1) = true \Rightarrow q_2^{cbr}(x_2) = true$ holds if $q_1 \preceq q_2$ and $dist(x_1,x_2) \le q_1.r - q_2.r$.*

Proof. The proof is symmetric to the proof of Theorem 3.9 (see Figure 4b for an illustration). That is, if x_1 is not an inlier by q_1, x_2 cannot be an inlier by q_2, either. □

Algorithm 1 Overall Procedure of MDUAL

Input: a data stream DS; an outlier query set Q_W at each window W
Output: a set O of outliers for each $q \in Q_W$ for each sliding window

1: $w \leftarrow$ the default window size; $s \leftarrow$ the default slide size; $c.size \leftarrow$ the default cell size;
2: **for each** window W of size w sliding by size s on DS and the current outlier query set Q_W **do**
3: $S_{exp} \leftarrow$ the expired slide; $S_{new} \leftarrow$ the new slide;
4: $Q_{exp} \leftarrow$ the expired outlier queries; $Q_{new} \leftarrow$ the new outlier queries;
5: $UGM_{prev} \leftarrow$ the previous unified group mapping;
6: $Q'_W \leftarrow \emptyset$; /* An initialized set of outlier queries valid in W */
7: **for each** $q_i \in Q_W$ **do**
8: $q_i.s' \leftarrow q_i.s' + s$; /* Accumulate slide size */
9: **if** $q_i.s' = q_i.s$ **then**
10: $q_i.s' = 0$ /* Initialize accumulated slide size */
11: $Q'_W \leftarrow$ add q_i;
12: **end if**
13: **end for**
14: /* 1. Net-Change Update: Algorithm 2 */
15: $UGM \leftarrow$ UG-Update($UGM_{prev}, S_{exp}, S_{new}, Q_{exp}, Q_{new}$);
16: CBRQ $\leftarrow$ *null* for all CBRQs in all $UG_Q^{ck} \in UGM$;
17: /* 2. Group-wise Coarse Processing: Algorithm 3 */
18: $\langle \mathbb{CBRQ}_{in}^{coar}, \mathbb{CBRQ}_{out}^{coar} \rangle \leftarrow$ GroupwiseCoarseProcessing(UGM);
19: /* 3. Group-wise Fine Processing: Algorithm 4 */
20: $\langle \mathbb{CBRQ}_{in}^{fine}, \mathbb{CBRQ}_{out}^{fine} \rangle \leftarrow$ GroupwiseFineProcessing(UGM);
21: $O \leftarrow$ the data points of the CBRQs in $\mathbb{CBRQ}_{out}^{coar} \cup \mathbb{CBRQ}_{out}^{fine}$;
22: **return** O;
23: **end for**

Evidently, a CBRQ whose result can be propagated to a larger number of other CBRQs should be processed with a higher priority. This requirement brings about two alternative prioritization strategies based on the two (inlier- and outlier-) propagation properties, as given in Definitions 3.11 and 3.12.

Definition 3.11. (Inlier-first prioritization) CBRQs are prioritized with higher preferences to *less restrictive* conditions so as to prefer a CBRQ which makes more other CBRQs return *false* by the inlier propagation property (Theorem 3.9). □

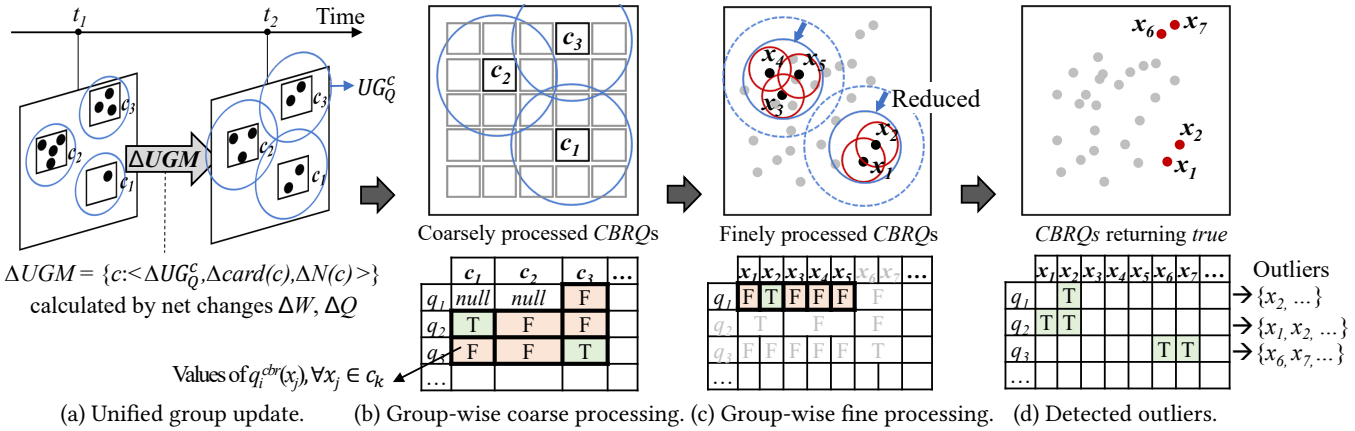
Definition 3.12. (Outlier-first prioritization) CBRQs are prioritized with higher preferences to *more restrictive* conditions so as to prefer a CBRQ which makes more other CBRQs return *true* by the outlier propagation property (Theorem 3.10). □

Typically, the number of inliers is much larger than the number of outliers, and therefore we adopt the *inlier-first prioritization* as the default strategy.

4 THE ALGORITHM MDUAL

4.1 Overview

Algorithm 1 outlines the overall procedure of MDUAL, and Figure 5 illustrates it. In every sliding window of data points and the current set of outlier queries from users, MDUAL first identifies the expired slide and the new slide, the expired outlier queries and the new outlier queries, and the previous UGM (Lines 3–5). Additionally, the outlier queries whose specified slide sizes become equal to the cumulative slide size from the last time of their evaluations are identified (Lines 6–13). Then, MDUAL conducts the following three



(a) Unified group update. (b) Group-wise coarse processing. (c) Group-wise fine processing. (d) Detected outliers.

Figure 5: Overall procedure of MDUAL.

steps. First, it updates *UGM* for the current window and outlier queries. The update is made efficiently by applying only the net-changes of data points and outlier queries between the consecutive windows to the previous *UGM* (Line 15 and Figure 5a). Second, given the updated *UGM*, it processes all the existing CBRQs in two steps: *group-wise coarse processing* and *group-wise fine processing*. In the coarse processing step, MDUAL processes some CBRQs in each unified group together by only using the triplet information in *UGM* (Line 18 and Figure 5b). Then, the remaining CBRQs in each unified group are individually inspected in the fine processing step by further pruning the triplet information (Line 20 and Figure 5c). This two-step group-wise approach allows MDUAL to process in each group as many CBRQs collectively as possible with *UGM*. Finally, MDUAL selects the CBRQs returning *true* and reports the data points of the CBRQs as outliers (Line 21 and Figure 5d). The *UGM* update step and the subsequent two group-wise processing steps are detailed in Sections 4.2 to 4.4.

Parameter adjustment: MDUAL requires the parameters for managing sliding windows and constructing unified groups. Given an initial set of outlier queries, the window size of a sliding window is set to the maximum of the window size conditions of all outlier queries in the set, the slide size to the greatest common divisor of the slide size conditions of all those outlier queries, and the cell size to the minimum of the distance threshold conditions of all those outlier queries. These values can be updated according to the changes of outlier query conditions during the algorithm execution.

4.2 Unified Group Update

MDUAL builds the *UGM* of the current window from the *UGM* of the previous window by incrementally updating it by the net-changes of data points and outlier queries between the two windows. Algorithm 2 outlines the procedure. It first counts the number of data points in each cell in the expired slide and in each cell in the new slide. Then, the net-changes of cell cardinality in each cell are calculated (Lines 2–4). Similarly, the expired CBRQs and the new CBRQs in each unified group are identified, and the net-changes of unified groups are calculated (Lines 6–8). Then, *UGM* is updated from the previous one based on those two net changes (Lines 10–17).

Algorithm 2 UG-Update

INPUT: the previous unified group mapping UGM_{prev}; the expired slide S_{exp}, the new slide S_{new}; the expired outlier queries Q_{exp}; the new outlier queries Q_{new}
OUTPUT: the current unified group mapping UGM

1: /* Net-change of cell cardinalities */
2: $\mathbb{CARD}_{exp} \leftarrow \{\langle c_k : |\{x_i \,|\, x_i \in c_k \wedge x_i \in S_{exp}\}|\rangle\}$;
3: $\mathbb{CARD}_{new} \leftarrow \{\langle c_k : |\{x_i \,|\, x_i \in c_k \wedge x_i \in S_{new}\}|\rangle\}$;
4: $\Delta\mathbb{CARD} \leftarrow \mathbb{CARD}_{new} - \mathbb{CARD}_{exp}$;
5: /* Net-change of unified groups */
6: $\mathbb{UG}_{exp} \leftarrow \{\langle c_k : \{\text{CBRQ of } q_i \text{ at } x_j \,|\, x_j \in c_k \wedge (x_j \in S_{exp} \vee q_i \in Q_{exp})\}\rangle\}$;
7: $\mathbb{UG}_{new} \leftarrow \{\langle c_k : \{\text{CBRQ of } q_i \text{ at } x_j \,|\, x_j \in c_k \wedge (x_j \in S_{new} \vee q_i \in Q_{new})\}\rangle\}$;
8: $\Delta\mathbb{UG} \leftarrow \mathbb{UG}_{new} - \mathbb{UG}_{exp}$;
9: /* The current unified group mapping */
10: $UGM \leftarrow UGM_{prev}$;
11: **for each** cell c_k in $\Delta\mathbb{UG}$ **do**
12: $UG_Q^{c_k}, card(c_k), N(c_k) \leftarrow$ the triplet associated with c_k in UGM;
13: /* Update the triplets */
14: $UG_Q^{c_k} \leftarrow UG_Q^{c_k} + \Delta\mathbb{UG}(c_k)$;
15: $card(c_k) \leftarrow card(c_k) + \Delta\mathbb{CARD}(c_k)$;
16: $N(c_k) \leftarrow N(c_k)+$ the changed neighbor cells of c_k by $\Delta\mathbb{UG}(c_k)$;
17: **end for**
18: **return** UGM;

4.3 Group-wise Coarse Processing

MDUAL processes a subset of CBRQs in a unified group collectively based on only the triplet in *UGM*. Specifically, for an outlier query q, MDUAL checks whether a cell c is an inlier cell or an outlier cell (see Definitions 4.1 and 4.2).

Definition 4.1. (INLIER CELL) A cell c is an *inlier cell* by an outlier query q if the CBRQs of q at all data points in c return false (i.e., $\forall x_j \in c, q^{cbr}(x_j)$ returns $false$). □

Definition 4.2. (OUTLIER CELL) A cell c is an *outlier cell* by an outlier query q if the CBRQs of q at all data points in c return true (i.e., $\forall x_j \in c, q^{cbr}(x_j)$ returns $true$). □

Identification of inlier/outlier cells: MDUAL classifies a cell into an inlier cell or an outlier cell by Theorems 4.3 or 4.4.

Algorithm 3 GroupwiseCoarseProcessing (Cell-Level)

INPUT: the unified group mapping UGM
OUTPUT: the pair $\langle \mathbb{CBRQ}^{coar}_{in}, \mathbb{CBRQ}^{coar}_{out} \rangle$ of identified CBRQs
1: $\mathbb{CBRQ}^{coar}_{in} \leftarrow \emptyset$; $\mathbb{CBRQ}^{coar}_{out} \leftarrow \emptyset$;
2: /* INLIER-FIRST PRIORITIZATION (DEFINITION 3.11) */
3: $Q'_W \leftarrow$ sort by the order of increasing $q.r$, $q.w$, and decreasing $q.k$;
4: **for each** cell c_k in UGM **do**
5: $UG^{c_k}_Q, card(c_k), N(c_k) \leftarrow$ the triplet associated with c_k in UGM;
6: **for each** $q_i \in Q'_W$ in a decreasing order of priority **do**
7: /* INLIER CELL CHECK (THEOREM 4.3) */
8: **if** c_k is an inlier cell by q_i **then**
9: /* $q_i^{cbr}(x_j) = false$ for all $x_j \in c_k$ */
10: $\mathbb{CBRQ}^{coar}_{in} \leftarrow$ add the CBRQs of q_i at each $x_j \in c_k$ and their inlier-propagated CBRQs (by Theorem 4.5);
11: /* OUTLIER CELL CHECK (THEOREM 4.4) */
12: **else if** c_k is an outlier cell by q_i **then**
13: /* $q_i^{cbr}(x_j) = true$ for all $x_j \in c_k$ */
14: $\mathbb{CBRQ}^{coar}_{out} \leftarrow$ add the CBRQs of q_i at each $x_j \in c_k$ and their outier-propagated CBRQs (by Theorem 4.6);
15: **end if**
16: **end for**
17: **end for**
18: **return** $\langle \mathbb{CBRQ}^{coar}_{in}, \mathbb{CBRQ}^{coar}_{out} \rangle$;

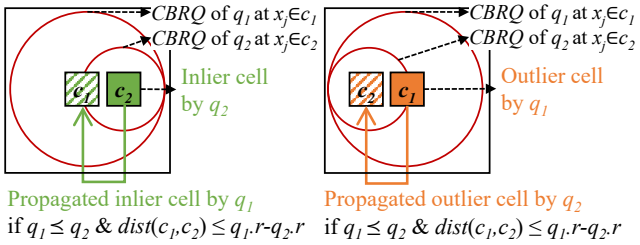
THEOREM 4.3. (INLIER CELL IDENTIFICATION) *A cell c is an inlier cell by an outlier query q if $q.r \geq c.size \wedge card(\{x \mid x \in c \wedge x \in \mathbb{W}^{q.s}_{q.w}\}) > q.k$.*

PROOF. Since the longest possible distance between two data points in c is its diagonal length $c.size$, all pairwise distances of data points in c are no longer than the distance threshold of q if the first condition (i.e., $q.r \geq c.size$) is satisfied. At the same time, since the set $\{x \mid x \in c \wedge x \in \mathbb{W}^{q.s}_{q.w}\}$ contains the data points in c that are included in the sliding window specified by q, all the data points have at least k neighbors in c if the second condition (i.e., $card(\{x \mid x \in c \wedge x \in \mathbb{W}^{q.s}_{q.w}\}) > q.k$) is satisfied as well. Thus, all of them are inliers by q. □

THEOREM 4.4. (OUTLIER CELL IDENTIFICATION) *A cell c is an outlier cell by an outlier query q if $\sum_{c' \in N'(c)} card(\{x \mid x \in c' \wedge x \in \mathbb{W}^{q.s}_{q.w}\}) \leq q.k$ where $N'(c) = \{c' \mid dist(c, c') \leq q.r + c.size\}$.*

PROOF. Given a data point x in a cell c, let another data point x' in another cell c' be a neighbor of x (i.e., $dist(x, x') \leq q.r$). The distance between x and the center of c, denoted as $dist(x, c)$, is at most $c.size/2$. By the triangle inequality, $dist(c, c') \leq dist(x, c) + dist(x, x') + dist(x', c') \leq c.size/2 + q.r + c.size/2 = q.r + c.size$. Then, the data points in $N'(c)$ contain all the neighbors of any data point in c. If the number of the data points in $N'(c)$ included in the sliding window specified by q is no more than the neighbor count threshold $q.k$, all the data points in c are outliers by q. □

The cells and triplets in UGM enable MDUAL to find inlier/outlier cells efficiently. Given a cell c, $card(c)$ is further inspected as in Theorem 4.3. At the same time, $N(c)$ is further inspected to find $N'(c)$ as in Theorem 4.4. It is guaranteed that $N'(c) \subseteq N(c)$ because by definition the distance threshold for finding $N(c)$ (i.e., $max_{q_i \in Q}(q_i.r) + c.size$) is no smaller than that for finding $N'(c)$ (i.e., $q.r + c.size$).



(a) Propagated inlier cell. (b) Propagated outlier cell.

Figure 6: Examples of propagated inlier/outlier cells.

Propagation of inlier/outlier cells: An inlier cell or an outlier cell by an outlier query indicates that all CBRQs of the outlier query at the data points in the cell have the same Boolean value. Thus, it is straightforward to extend the propagation properties in Theorems 3.9 and 3.10 to the cell level by replacing the distance conditions (i.e., $dist(x_1, x_2)$) between two data points in the properties by those between two cells (i.e., $dist(c_1, c_2)$). The extended cell-level properties are given in Theorems 4.5 and 4.6. Figure 6 shows examples of propagated inlier or outlier cells. Note that MDUAL maximizes the propagation opportunities by following the inlier-first prioritization in Definition 3.11 and also efficiently propagates inlier/outlier cells by considering only the cells in $N(c)$.

THEOREM 4.5. (INLIER-CELL PROPAGATION PROPERTY) *Given two cells c_1 and c_2 and two outlier queries q_1 and q_2, if c_2 is an inlier cell by q_2, then c_1 is an inlier cell by q_1 when $q_1 \preceq q_2$ and $dist(c_1, c_2) \leq q_1.r - q_2.r$.*

PROOF. As shown in Figure 6a, if all $x_2 \in c_2$ are inliers by q_2, all $x_1 \in c_1$ are inliers by q_1 when $q_1 \preceq q_2$ and $dist(c_1, c_2) \leq q_1.r - q_2.r$, because they are guaranteed to have at least $q_1.k$ neighbors. Hence, if c_2 is an inlier cell by q_2, c_1 must be an inlier cell by q_1, too. □

THEOREM 4.6. (OUTLIER-CELL PROPAGATION PROPERTY) *Given two cells c_1 and c_2 and two outlier queries q_1 and q_2, if c_1 is an outlier cell by q_1, then c_2 is an outlier cell by q_2 when $q_1 \preceq q_2$ and $dist(c_1, c_2) \leq q_1.r - q_2.r$.*

PROOF. The proof is symmetric to the proof of Theorem 4.5 (see Figure 6b for an illustration). □

Algorithm 3 outlines the procedure of the group-wise coarse processing. First, two sets of CBRQs are initialized, one for inliers ($\mathbb{CBRQ}^{coar}_{in}$) and one for outliers ($\mathbb{CBRQ}^{coar}_{out}$) (Line 1), and outlier queries are sorted by the inlier-first prioritization strategy (Line 3). Then, for each cell (Line 4), if a cell is determined to be either an inlier cell or an outlier cell by an outlier query, it identifies all the corresponding CBRQs, which return $false$ for an inlier cell and $true$ for an outlier cell, and obtains their propagated CBRQs; and adds them to the corresponding processed CBRQ set (Lines 7–15). Finally, the pair of the processed CBRQ sets is returned (Line 18).

4.4 Group-wise Fine Processing

In this step, MDUAL individually processes the CBRQs not processed in the previous step; it identifies those CBRQs by inspecting their conditions based on the refined triplet in UGM. Algorithm 4 outlines the overall procedure. MDUAL first tightens the scope of a

Algorithm 4 GroupwiseFineProcessing (Point-Level)

INPUT: the unified group mapping UGM
OUTPUT: the pair $\langle \mathbb{CBRQ}_{in}^{fine}, \mathbb{CBRQ}_{out}^{fine} \rangle$ of identified CBRQs

1: $\mathbb{CBRQ}_{in}^{fine} \leftarrow \emptyset$; $\mathbb{CBRQ}_{out}^{fine} \leftarrow \emptyset$;
2: **for each** cell c_k in UGM **do**
3: $UG_Q^{c_k}, card(c_k), N(c_k) \leftarrow$ the triplet associated with c_k in UGM;
4: $RUG_Q^{c_k} \leftarrow$ a group of unidentified CBRQs reduced from $UG_Q^{c_k}$;
5: $RN(c_k) \leftarrow$ neighbor cells of c_k reduced from $N(c_k)$ by $RUG_Q^{c_k}$;
6: **for each** CBRQ of q_i at x_j in $RUG_Q^{c_k}$ **do**
7: $nn \leftarrow |\{x_k \,|\, x_k \in RN(c_k) \wedge dist(x_j, x_k) \leq q_i.r \wedge x_k \in \mathbb{W}_{q_i.w}^{q_i.s}\}|$;
8: **if** $nn \geq q_i.k$ **then**
9: /* $q_i^{cbr}(x_j) = false$ */
10: $\mathbb{CBRQ}_{in}^{fine} \leftarrow$ add the CBRQ and its inlier-propagated CBRQs (by Theorem 3.9);
11: **else**
12: /* $q_i^{cbr}(x_j) = true$ */
13: $\mathbb{CBRQ}_{out}^{fine} \leftarrow$ add the CBRQ and its outlier-propagated CBRQs (by Theorem 3.10);
14: **end if**
15: **end for**
16: **end for**
17: **return** $\langle \mathbb{CBRQ}_{in}^{fine}, \mathbb{CBRQ}_{out}^{fine} \rangle$;

unified group. Specifically, each unified group is reduced to a set of the unidentified CBRQs in the group, and accordingly the neighboring cells are reduced as well by the new maximal expanded range query of the reduced unified group (Lines 2–5). This reduction is illustrated in Figure 5c; as the CBRQs of the outlier queries q_2 and q_3 at all data points have been processed in the previous step, each unified group is reduced to include only the CBRQs of the outlier query q_1 which remains unidentified. For each CBRQ of an outlier query q_i at a data point x_j in each reduced unified group (Line 6), it finds the neighbors of x_j by q_i from the reduced neighboring cells (Line 7), confirms that the CBRQ of q_i and its propagated CBRQs all return $false$ (if inliers) or $true$ (if outliers), and adds them to the corresponding processed CBRQ set ($\mathbb{CBRQ}_{in}^{fine}$ for inliers or $\mathbb{CBRQ}_{out}^{fine}$ for outliers) (Lines 8–14). Finally, the pair of the CBRQ sets processed in this step is returned (Line 17).

4.5 Complexity Analysis

The worst-case time and space complexities of MDUAL are given by Theorems 4.7 and 4.8, respectively, which are linear to the total number of data points.

THEOREM 4.7. *Given the number N_Q of outlier queries, the number N_D of data points in a sliding window, the number N_G of unified groups, and the number N_C of unidentified CBRQs in the group-wise coarse processing, the time complexity of MDUAL is $O(N_Q N_G^2 + N_D(N_Q + N_C/N_G))$.*

PROOF. The time complexity of updating UGM is $O(N_D + N_D N_Q + N_G^2)$ because indexing a data point into a cell takes constant time, creating CBRQs is needed for all pairs of data points and outlier queries, and finding neighboring cells requires pair-wise distance computations between cells. The time complexity of group-wise coarse processing is $O(N_Q N_G^2)$ for identifying inlier cells and outlier cells. The time complexity of group-wise fine processing is $O(N_D N_C/N_G)$, because processing each of the N_C unidentified CBRQs requires to visit the data points in a neighbor cell whose count is N_D/N_G on average. Thus, the total time complexity is $O(N_D + N_D N_Q + N_G^2 + N_Q N_G^2 + N_D N_C/N_G) = O(N_Q N_G^2 + N_D(N_Q + N_C/N_G))$. □

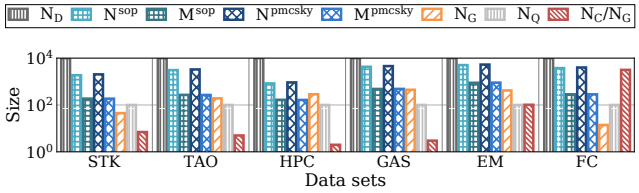


Figure 7: Comparison of the sizes of the complexity variables in real data sets (y-axis is in log scale).

THEOREM 4.8. *Given the number N_Q of outlier queries, the number N_D of data points in a sliding window, the number N_G of unified groups, the space complexity of MDUAL is $O(N_Q + N_D + N_G)$.*

PROOF. The space complexities of managing outlier queries, data points in sliding windows, and UGM are $O(N_Q)$, $O(N_D)$, and $O(N_G)$, respectively. Thus, the total space complexity is to $O(N_Q + N_D + N_G)$. □

Let us compare the time complexity and the space complexity of MDUAL with those of two state-of-the-art algorithms, SOP [6] and pMCSKY [35]. Basically, given a set of data points and a set of outlier queries, the two algorithms first find all the neighbors of each data point required for processing all outlier queries, called the *skyband points*, and evaluate each outlier query for each data point using them. The time complexities of SOP and pMCSKY are $O(N_D(N^{sop} + N_Q M^{sop}))$ and $O(N_D(N^{pmcsky} + N_Q M^{pmcsky}))$, respectively, where N^{sop} and N^{pmcsky} are the number of operations for searching the skyband points by each algorithm and M^{sop} and M^{pmcsky} are the sizes of the skyband points in each algorithm. The space complexity of SOP is $O(N_Q + N_D M^{sop})$ and that of pMCSKY is $O(N_Q + N_D M^{pmcsky})$. Since the skyband points can include all data points in the worst case, the sizes of N^{sop}, N^{pmcsky}, M^{sop}, and M^{pmcsky} can increase up to N_D. MDUAL has a lower time complexity and a lower space complexity than SOP and pMCSKY in practice because typically $N_D > N^{sop} \approx N^{pmcsky} > M^{sop} \approx M^{pmcsky} > N_G > N_Q > N_C/N_G$ in real data sets (except the FC data set), as shown in Figure 7.

5 EXPERIMENTS

Thorough experiments demonstrate that MDUAL outperforms existing algorithms.

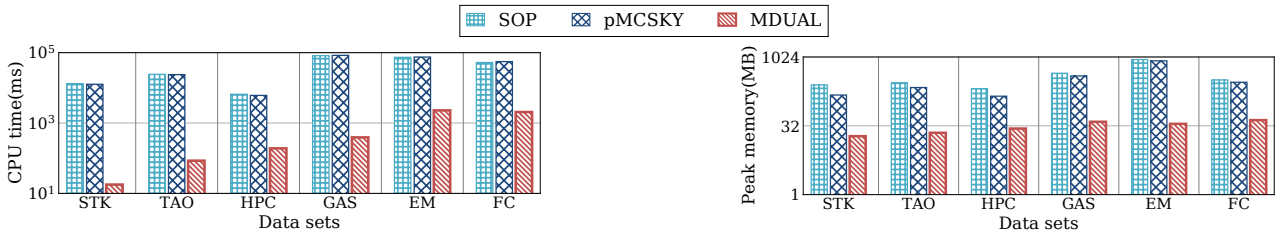
- MDUAL's duality-based unified processing achieves orders of magnitude smaller CPU time and peak memory than the state-of-the-art algorithms (Section 5.2).
- MDUAL's performance is far more robust to the multiplicity of queries and the dynamicity of queries than the state-of-the-art algorithms (Section 5.3).
- MDUAL's performance is generalizable to diverse outlier queries with different distributions of outlier query condition values (Section 5.4).

Table 3: Data sets and the default outlier query conditions.

Data set	Dim.	Num. of data points	$q.r$	$q.k$	$q.w$	$q.s$
STK [29]	1	1.1 million	0.5	5	1000	50
TAO [28]	3	0.6 million	1.5	5	1000	50
HPC [12]	7	1.0 million	10	5	1000	50
GAS [19]	10	0.9 million	1.5	5	1000	50
EM [13]	16	1.0 million	115	5	1000	50
FC [12]	55	0.6 million	525	5	1000	50

Table 4: Dynamicity and multiplicity of an outlier query set.

Parameter	Range (default)
Number of outlier queries	1, 10, 100, 1000
Change rate of outlier queries	0.0, 0.2, 0.4, 0.8. 1.0
Conditions varied	All of $q.r$, $q.k$, $q.w$, and $q.s$
Condition value distribution (d = the default value)	Uniform([$[1d, 5d]$, $[1d, 10d]$, $[1d, 20d]$]), Gaussian($\mu = [2d, 4d, 8d]$, $\sigma = 1d$), Exponential($\mu = [2d, 4d, 8d]$)



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

Figure 8: Overall performance comparison between MDUAL and the two state-of-the-art algorithms (y-axes are in log scale).

5.1 Experiment Setup

Data sets: The six real data sets (see Table 3) are the benchmark data sets commonly used in other relevant work [6, 35, 36]. STK [29] contains one-dimensional stock trading transaction records. TAO [28] contains three-dimensional oceanographic records from the Tropical Atmosphere Ocean project. HPC [12] contains seven-dimensional electric power consumption data, and GAS [19] contains ten-dimensional household gas sensor data. EM [13] and FC [12] contain high-dimensional chemical sensor data and forest cover type data, respectively. Table 3 shows the profiles (dimensionality and size) of the data sets.

Outlier query sets: An outlier query has four conditions: the distance threshold, the neighbor count threshold, the window size, and the slide size. The default values of each condition for each data set are summarized in Table 3; these values have been chosen to make the ratio of outliers to data points in a window to around 1% as suggested by Tran et al. [36]. To simulate outlier queries reflecting different degrees of multiplicity and dynamicity, for each data set we generated outlier queries according to the parameter specifications summarized in Table 4; the number of queries is varied from 1 to 1,000 following the related work [6, 35]. Unless stated otherwise, all parameters in Table 4 are set to their default values in each experiment. For instance, a default outlier query set for the STK data set consists of 100 outlier queries whose condition values are uniformly sampled from the range between 1 to 10 times of their default values, i.e., $q.r \sim$Uniform([0.5, 5]), $q.k \sim$Uniform([5, 50]), $q.w \sim$Uniform([1000, 10000]), and $q.s \sim$Uniform([50, 500]), and 20% of outlier queries in the outlier query set are replaced in every sliding window.

Algorithms: The two state-of-the-art MD-DODDS algorithms, SOP [6] and pMCSKY [35], are compared with MDUAL. They focus on optimizing finding a set of neighbors defined by multiple outlier queries for *individual* data points, as will be explained in more detail in Section 6. To assess the unified grouping strategy, we also compare MDUAL with two variants: MDUAL-D and MDUAL-Q where only the data similarity (former) and only the query similarity (latter) are exploited when grouping CBRQs. For the sake of fair comparison, we implemented all algorithms in JAVA. The source code of MDUAL is publicly available [42].

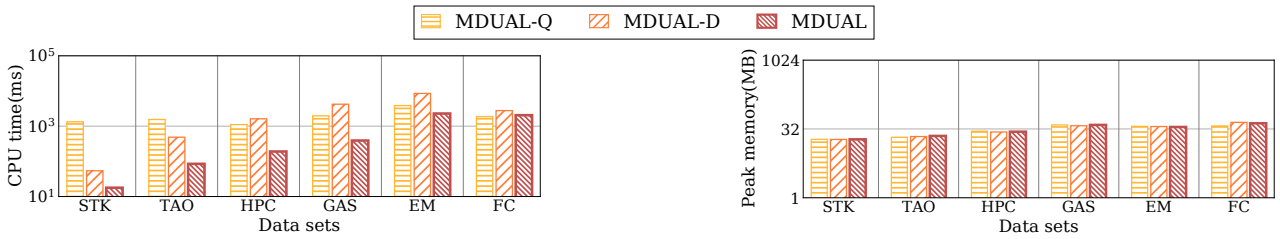
Computing platform: We conducted all experiments on a Linux server with Intel Xeon Silver 4116 (2.10GHz CPU), 64GB RAM, and 26TB HDD. Ubuntu 16.04.6 LTS and JDK 1.8.0 252 were installed on the server.

Performance metrics: Reducing the latency and memory consumption is particularly important in the multi-query processing environment. In this regard, the performance metrics used are the *CPU time* and the *maximum memory* consumed (a.k.a. *peak memory*) to detect all outliers for all outlier queries in every window. Specifically, the JAVA ThreadMXBean interface and a separate thread are used to record CPU time and peak memory, respectively, as done by Tran et al. [36]. The average of *five* repetitions in each test is reported as a result. Note that the accuracy is not considered because all the compared algorithms are *exact* algorithms.

5.2 Efficacy of Duality-based Unified Processing

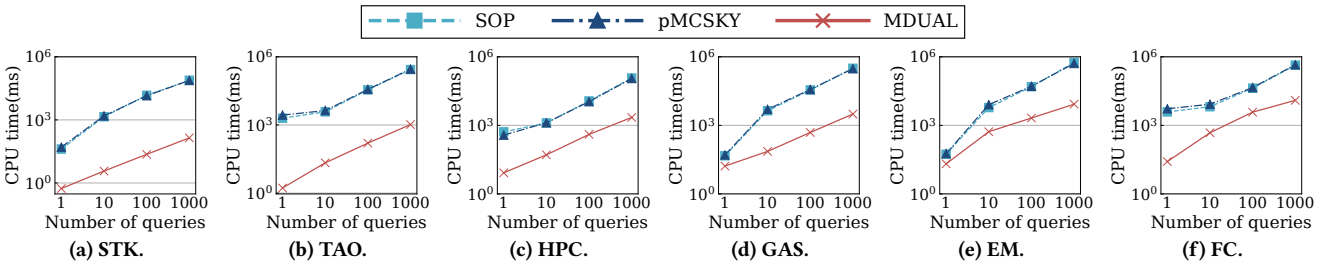
In order to evaluate the efficacy of duality-based unified processing, we compare the overall performances and then provide further analysis with regard to the grouping strategy discussed in Section 3.1 and the prioritization strategy discussed in Section 3.2.

Overall performance: Figure 8 shows the overall performance results of the three algorithms (SOP, pMCSKY, and MDUAL) for each of the six real data sets; note the logarithmic scale of the CPU time and the peak memory. MDUAL was by far the fastest, outspeeding SOP by 221 times and pMCSKY by 216 times when averaged over all data sets; besides, MDUAL consumed only 31MB of peak memory when averaged over all data sets, which was 13 times less than 401MB of SOP and 11 times less than 340MB of pMCKSY. This better performance of MDUAL demonstrates the effectiveness of duality-based unified processing for detecting outliers with multiple and dynamic outlier queries. While SOP and pMCSKY process multiple outlier queries separately for each data point, MDUAL



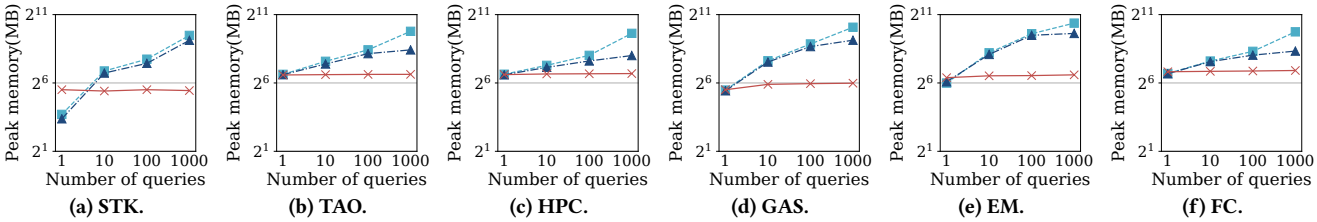
(a) CPU time. (b) Peak memory.

Figure 9: Performance comparison of the MDUAL variants (y-axes are in log scale).



(a) STK. (b) TAO. (c) HPC. (d) GAS. (e) EM. (f) FC.

Figure 10: CPU time when varying the number of queries (both x- and y-axes are in log scale).



(a) STK. (b) TAO. (c) HPC. (d) GAS. (e) EM. (f) FC.

Figure 11: Peak memory when varying the number of queries (both x- and y-axes are in log scale).

processes data-query pairs collectively as a group and reduces the computation for outlier detection with the prioritization strategy. Besides, unlike SOP and pMCSKY, the efficient net-change-based update of *UGM* enables MDUAL to process dynamic outlier queries incrementally instead of repeatedly from scratch.

Grouping strategy: Figure 9 shows the performance results of the MDUAL variants: MDUAL-D and MDUAL-Q. Compared with SOP and pMCSKY, all MDUAL variants consistently resulted the smaller CPU time and peak memory in all data sets, which demonstrate the effectiveness of the grouped processing based on the integration of data and outlier queries. The data similarity varies for each data set; in lower-dimensional data sets (STK and TAO) the grouping based on data-similarity was more effective than the grouping based on query-similarity, while the opposite was observed in higher-dimensional data sets (HPC, GAS, EM, and FC). Regardless, the *unified* grouping of MDUAL showed the best performance for most data sets by exploiting both data similarity and query similarity. (FC was an exception in that MDUAL-Q slightly outperformed MDUAL, but the difference was negligible.)

Prioritization strategy: We applied the two types of prioritization strategies, inlier-first and outlier-first, and measured the propagation ratio of CBRQs to examine how effective the prioritization strategy was. The *propagation ratio* is calculated as the ratio of the number of CBRQs propagated through the coarse and fine group-wise processing steps to the total number of CBRQs created in all sliding windows. The higher the propagation ratio is, the more computation is saved. As shown in Table 5, the inlier-first strategy resulted in the propagation ratio of 73.65% averaged over all data sets, which means that approximately only a quarter of all CBRQs are directly evaluated by using the triplet in *UGM*. On the other hand, the outlier-first strategy resulted in the propagation ratio of only 0.03% averaged over all data sets. As expected from the fact that the ratio of outliers to inliers is very small (i.e., 1%) (by the definition of an outlier), the inlier-first strategy, adopted by MDUAL as the default, is confirmed to be much more effective than the outlier-first strategy.

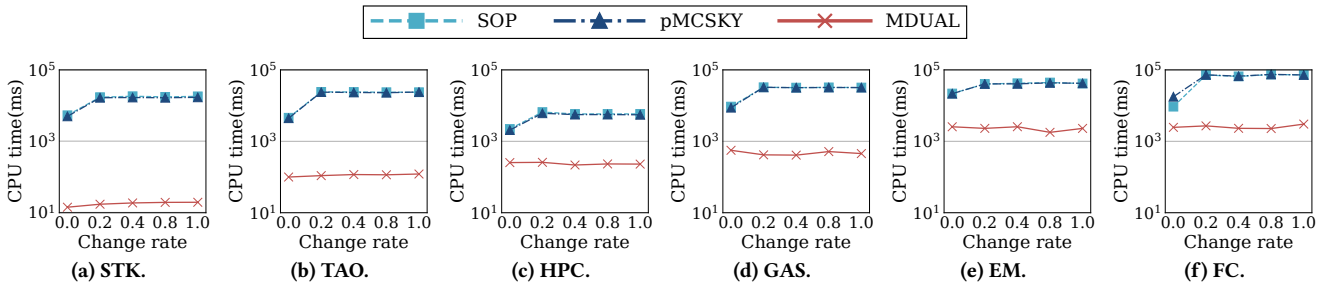
Table 5: Ratio of the number of the propagated CBRQs to the total number of CBRQs by the two prioritization strategies.

	STK	TAO	HPC	GAS	EM	FC
Inlier-first	79.66%	76.13%	82.97%	72.28%	65.85%	64.98%
Outlier-first	0.06%	0.09%	0.01%	0.02%	0.00%	0.00%

5.3 Robustness to Multiplicity and Dynamicity

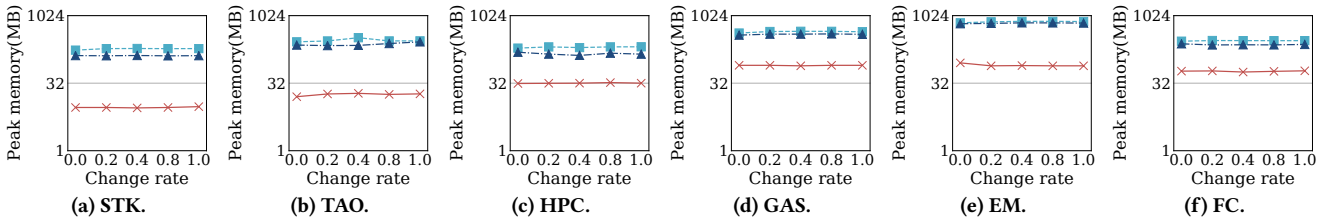
We evaluate the robustness of MDUAL to the multiplicity of queries characterized by the number of outlier queries and the dynamicity of queries characterized by the change rate of an outlier query set.

Multiplicity (number of outlier queries): Figures 10 and 11 respectively show the CPU time and peak memory results of the



(a) STK. (b) TAO. (c) HPC. (d) GAS. (e) EM. (f) FC.

Figure 12: CPU time when varying the change rate of a query set (y-axes are in log scale).



(a) STK. (b) TAO. (c) HPC. (d) GAS. (e) EM. (f) FC.

Figure 13: Peak memory when varying the change rate of a query set (y-axes are in log scale).

three algorithms when the number of outlier queries is varied from 1 to 1,000. Evidently, the CPU time increases with the number of outlier queries in all three algorithms. However, MDUAL was consistently the fastest, and the rate of increase was the smallest—368 times from one outlier query to 1,000 outlier queries averaged over all data sets, compared with 3,255 times for SOP and 2,778 times for pMCSKY. Thus, MDUAL is clearly shown to handle multiple outlier queries more efficiently than the other two. MDUAL's peak memory was relatively constant (79MB–87MB, averaged over all data sets) despite the increasing number of outlier queries, because the size of *UGM*, the main data structure used in MDUAL, is determined by the number of cells, which is independent of the number of outlier queries. This result is clearly contrasted with the linear increase in the other algorithms (70MB–940MB for SOP and 69MB–470MB for pMCSKY), because the size of the data structure (called "L-Sky") for managing the skyband points used by SOP and pMCSKY is highly dependent on the number of outlier queries.

Dynamicity (change rate of an outlier query set): Figures 12 and 13 respectively show the CPU time and peak memory of the three algorithms when the change rate of an outlier query set between consecutive windows is varied from 0.0 (not changed) to 1.0 (entirely changed). When there was no change in the outlier query set, MDUAL's CPU time was smaller than SOP by 77 times and pMCSKY by 72 times, averaged over all data sets. Then, when an outlier query set was changed, the CPU time increased significantly for SOP and pMCSKY, but not so much for MDUAL; as a result, the gap between MDUAL and SOP was grown to 217 times and that between MDUAL and pMCSKY was grown to 207 times. The reason is that SOP and pMSKY do not adapt to the change in the outlier query set but always process all outlier queries in the new outlier query set from scratch, whereas MDUAL does adapt by updating *UGM* with only the net-change of outlier queries between windows, not affecting the overall processing time significantly. The peak memory was constant for all three algorithms, where MDUAL's peak memory was 7 times smaller than the other two algorithms, averaged over all cases. The reason is that the main factors affecting the memory consumption are the number of outlier queries and the number of data points for all algorithms and additionally the number of cells for MDUAL; all of them are independent of the change rate of an outlier query set.

5.4 Generalizability to Diverse Outlier Queries

Generalizability of MDUAL's performance across diverse outlier queries, with different or evolving distributions of the outlier query condition values, is of practical importance in a multiple dynamic outlier query processing environment. To this end, we generated the outlier query condition values from the Gaussian and Exponential distributions as well as the Uniform (default) distribution, with the distribution parameters set as specified in Table 4. Figure 14 shows in a scatter plot the resulting CPU time and peak memory of each of the three algorithms for each of the three distributions. The color of a symbol indicates the distribution from which the outlier query condition values are created, and for each pair of a distribution and an algorithm, three results (i.e., symbols) correspond to three distribution parameter values in Table 4. MDUAL consistently achieved the smallest CPU time and peak memory among all algorithms regardless of the distribution. Concretely, MDUAL was 22 times faster and consumed 10 times less memory than the other algorithms when averaged over all pairs of a distribution and a data set. This result shows that the effectiveness of duality-based unified processing of MDUAL is not specific to a certain distribution of outlier query condition values but is generalizable to different distributions.

6 RELATED WORK

The problem of outlier detection has been actively studied over the past decades [4]. Given the various approaches for defining outliers such as distance-based [20], density-based [5], cluster-based [17], or tree-based [26], adopting those approaches into a streaming setting has received a lot of attention recently [11, 16, 32, 41]. Among

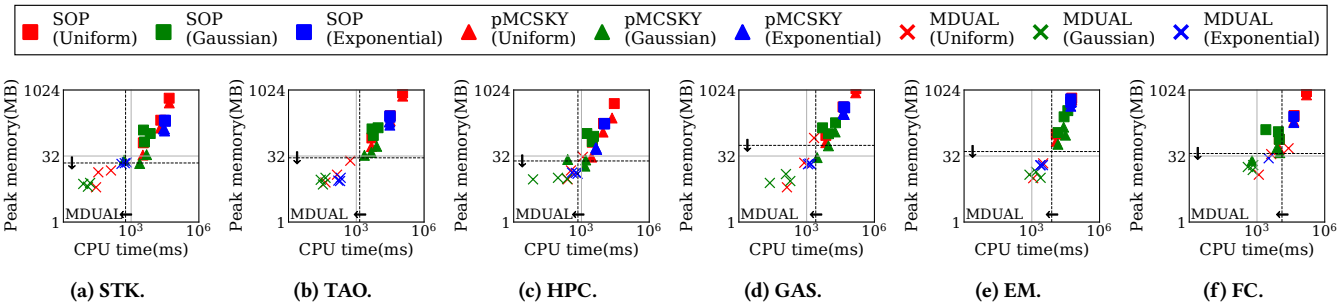


Figure 14: CPU time and peak memory when varying the distribution of query condition values (both axes are in log scale).

them, the distance-based outlier detection approach has been very popular due to its intuitive definition on outliers and high usability in an unsupervised setting. Accordingly, a streaming version of the distance-based outlier detection problem, which we call DODDS (see Definition 2.1), is a well-studied problem.

The existing DODDS algorithms [3, 9, 21, 39, 40] commonly follow three steps in every sliding window [36]: (1) process the expired slide, (2) process the new slide, and (3) identify outliers. Each of these algorithms optimizes the repetition of the three steps; for example, MCOD [21] and LEAP [9] respectively employ *micro-cluster-based pruning* and *minimal probing principle* to reduce the neighbor search time in subsequent sliding windows. NETS [40] employs *set-based processing* to further improve the speed by saving redundant operations between the first and second steps. All of them, however, are meant for a *single* outlier query, not multiple (not to mention dynamic) outlier queries.

MD-DODDS (see Definition 2.2) is a multi-query version of the DODDS problem. A naive approach for MD-DODDS would be to run a DODDS algorithm separately for every outlier query, but it suffers from inefficient processing because the neighbors of a data point for distinct but similar outlier queries can overlap significantly. Thus, AMCOD [22], SOP [6], and pMCSKY [35] each addressed the redundancy issue by using the skyline query technique [30]. Specifically, for each data point, they try to find neighbor candidates, called the "skyband points," and use them to process all outlier queries without actual neighbor searches. Details of the algorithms are as follows:

- AMCOD groups a part of data points into micro-clusters so that the data points inside the micro-clusters are guaranteed to be inliers for every outlier query. The skyband points are retrieved only for those outside the micro-clusters. AMCOD allows only the distance threshold and neighbor count threshold conditions to vary, i.e., not the window size and slide size conditions.
- SOP minimizes the number of data points examined to process outlier queries, by *prioritizing* the skyband points. For a data point, its neighbor in the skyband points has a higher priority if it is newer and closer to the data point than the others. While the prioritization reduces the number of skyband points to handle, the overlapping skyband points of close data points are not recognized and are handled redundantly.
- pMCSKY improves AMCOD by adopting the prioritization of the skyband points introduced in SOP. Only the data points outside micro-clusters are considered for searching and prioritizing the skyband points. However, the skyband points of each data point are still handled separately without considering the overlaps.

Limitation of the existing MD-DODDS algorithms: The existing algorithms address the multiplicity challenge only *partially*, by optimizing the maintenance of the neighbor sets determined by multiple outlier queries for each data point. More concretely they partially exploit the query similarity by merging the neighbor sets returned by each outlier query; their consideration of the data similarity is limited as well: AMCOD and pMCSKY consider it only for the data points guaranteed to be inliers inside the micro-clusters, and SOP does not consider it at all. Most importantly, none of them fully exploits both the data similarity and the query similarity in a unified way as MDUAL does with the use of duality-based unified processing. Furthermore, SOP and pMCSKY assume that the initially given set of outlier queries is fixed. Therefore, they are not able to incrementally adapt to a changing set of outlier queries, being incapable of addressing the dynamicity challenge.

7 CONCLUSION

This paper proposed a distance-based data-stream outlier detection algorithm, abbreviated as **MDUAL**, that addresses a new challenge—multiplicity and dynamicity of queries. In recognition of the duality of data and queries in the outlier detection process, the idea of *duality-based unified processing* was devised to remove redundant, repetitive computations in processing multiple dynamic outlier queries on a data stream. By virtue of data-query grouping and prioritized group processing, MDUAL can efficiently adapt to the dynamic change of outlier queries and effectively prunes the search space for outlier detection. Experiments showed that MDUAL performed 216 to 221 times faster while consuming 11 to 13 times less main memory than the state-of-the-art algorithms, averaged over six real data sets. We believe that MDUAL enables effective real-time outlier detection in many applications facing the challenge of query multiplicity and dynamicity.

ACKNOWLEDGMENTS

This work was partly supported by Samsung Electronics Co., Ltd. (IO201211-08051-01) through the Strategic Collaboration Academic Program and Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00862, DB4DL: High-Usability and Performance In-Memory Distributed DBMS for Deep Learning).

REFERENCES

[1] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. 2016. Fraud Detection System: A Survey. *Journal of Network and Computer Applications* 68 (2016), 90–113.

[2] Charu C Aggarwal. 2016. *Outlier Analysis* (2 ed.). Springer.

[3] Fabrizio Angiulli and Fabio Fassetti. 2007. Detecting Distance-based Outliers in Streams of Data. In *Proceedings of the 2007 ACM Conference on Information and Knowledge Management*. 811–820.

[4] Azzedine Boukerche, Lining Zheng, and Omar Alfandi. 2020. Outlier Detection: Methods, Models, and Classification. *Comput. Surveys* 53, 3 (2020), 1–37.

[5] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: Identifying Density-based Local Outliers. *ACM SIGMOD Record* 29, 2 (2000), 93–104.

[6] Lei Cao, Jiayuan Wang, and Elke A Rundensteiner. 2016. Sharing-aware Outlier Analytics over High-volume Data Streams. In *Proceedings of the 2016 ACM International Conference on Management of Data*. 527–540.

[7] Lei Cao, Qingyang Wang, and Elke A Rundensteiner. 2014. Interactive Outlier Exploration in Big Data Streams. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1621–1624.

[8] Lei Cao, Mingrui Wei, Di Yang, and Elke A Rundensteiner. 2015. Online Outlier Exploration over Large Datasets. In *Proceedings of the 2015 ACM International Conference on Knowledge Discovery and Data Mining*. 89–98.

[9] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A Rundensteiner. 2014. Scalable Distance-based Outlier Detection over High-volume Data Streams. In *Proceedings of the 2014 IEEE International Conference on Data Engineering*. 76–87.

[10] Zhida Chen, Gao Cong, Zhenjie Zhang, Tom ZJ Fuz, and Lisi Chen. 2017. Distributed Publish/Subscribe Query Processing on the Spatio-Textual Data Stream. In *Proceedings of the 2017 IEEE International Conference on Data Engineering*. 1095–1106.

[11] Andrew Cook, Göksel Mısırlı, and Zhong Fan. 2019. Anomaly Detection for IoT Time-series Data: A Survey. *IEEE Internet of Things Journal* (2019).

[12] Dheeru Dua and Casey Graff. 2019. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. Accessed: 2020-07-01.

[13] Jordi Fonollosa, Sadique Sheik, Ramón Huerta, and Santiago Marco. 2015. Reservoir Computing Compensates Slow Response of Chemosensor Arrays Exposed to Fast Varying Gas Concentrations in Continuous Monitoring. *Sensors and Actuators B: Chemical* 215 (2015), 618–629.

[14] Peng Gao, Xusheng Xiao, Ding Li, Zhichun Li, Kangkook Jee, Zhenyu Wu, Chung Hwan Kim, Sanjeev R Kulkarni, and Prateek Mittal. 2018. SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection. In *Proceedings of the 27th USENIX Security Symposium*. 639–656.

[15] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. In *Proceedings of the 2015 ACM International Conference on Management of Data*. 1917–1923.

[16] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. 2013. Outlier Detection for Temporal Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (2013), 2250–2267.

[17] Zengyou He, Xiaofei Xu, and Shengchun Deng. 2003. Discovering Cluster-based Local Outliers. *Pattern Recognition Letters* 24, 9-10 (2003), 1641–1650.

[18] Bastian Hoßbach and Bernhard Seeger. 2013. Anomaly Management Using Complex Event Processing: Extending Data Base Technology Paper. In *Proceedings of the 2013 International Conference on Extending Database Technology*. 149–154.

[19] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, Nikolai F Rulkov, and Irene Rodriguez-Lujan. 2016. Online Decorrelation of Humidity and Temperature in Chemical Sensors for Continuous Monitoring. *Chemometrics and Intelligent Laboratory Systems* 157 (2016), 169–176.

[20] Edwin M Knox and Raymond T Ng. 1998. Algorithms for Mining Distance-based Outliers in Large Datasets. In *Proceedings of the 1998 International Conference on Very Large Data Bases*. 392–403.

[21] Maria Kontaki, Anastasios Gounaris, Apostolos N Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2011. Continuous Monitoring of Distance-based Outliers over Data Streams. In *Proceedings of the 2011 IEEE International Conference on Data Engineering*. 135–146.

[22] Maria Kontaki, Anastasios Gounaris, Apostolos N Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2016. Efficient and Flexible Algorithms for Monitoring Distance-based Outliers over Data Streams. *Information Systems* 55 (2016), 37–53.

[23] Flip Korn, Shanmugavelayutham Muthukrishnan, and Yihua Wu. 2006. Modeling Skew in Data Streams. In *Proceedings of the 2006 ACM International Conference on Management of Data*. 181–192.

[24] SangJeong Lee, Youngki Lee, Byoungjip Kim, Kasim Selçuk Candan, Yunseok Rhee, and Junehwa Song. 2011. High-performance Composite Event Monitoring System Supporting Large Numbers of Queries and Sources. In *Proceedings of the 2011 ACM International Conference on Distributed Event-based System*. 137–148.

[25] Yuhang Lin, Byung Suk Lee, and Daniel Lustgarten. 2018. Continuous Detection of Abnormal Heartbeats from ECG Using Online Outlier Detection. In *Proceedings of the 5th International Conference on Information Management and Big Data*. 349–366.

[26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the 2008 IEEE International Conference on Data Mining*. 413–422.

[27] Ahmed R Mahmood, Anas Daghistani, Ahmed M Aly, Mingjie Tang, Saleh Basalamah, Sunil Prabhakar, and Walid G Aref. 2018. Adaptive Processing of Spatial-Keyword Data over a Distributed Streaming Cluster. In *Proceedings of the 2018 ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 219–228.

[28] NOAA. 2020. Tropical Atmosphere Ocean Project. https://www.pmel.noaa.gov. Accessed: 2020-07-01.

[29] University of Pennsylvania. 2020. Wharton Research Data Services. https://wrds-web.wharton.upenn.edu/wrds/. Accessed: 2020-07-01.

[30] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.

[31] Ramsey M Raafat, Nick Chater, and Chris Frith. 2009. Herding in Humans. *Trends in Cognitive Sciences* 13, 10 (2009), 420–428.

[32] Mahsa Salehi and Lida Rashidi. 2018. A Survey on Anomaly detection in Evolving Data. *ACM SIGKDD Explorations Newsletter* 20, 1 (2018), 13–23.

[33] Yooju Shin, Susik Yoon, Patara Trirat, and Jae-Gil Lee. 2019. CEP-Wizard: Automatic Deployment of Distributed Complex Event Processing. In *Proceedings of the 2019 IEEE International Conference on Data Engineering*. 2004–2007.

[34] Jordan Tigani and Siddartha Naidu. 2014. *Google BigQuery Analytics*. John Wiley & Sons.

[35] Theodoros Toliopoulos and Anastasios Gounaris. 2019. Multi-Parameter Streaming Outlier Detection. In *Proceedings of the 2019 IEEE/WIC/ACM International Conference on Web Intelligence*. 208–216.

[36] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based Outlier Detection in Data Streams. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1089–1100.

[37] Simon Walton, Eamonn Maguire, and Min Chen. 2014. Multiple Queries with Conditional Attributes (QCATs) for Anomaly Detection and Visualization. In *Proceedings of the 11th Workshop on Visualization for Cyber Security*. 17–24.

[38] Jarrod West and Maumita Bhattacharya. 2016. Intelligent Financial Fraud Detection: A Comprehensive Review. *Computers & Security* 57 (2016), 47–66.

[39] Di Yang, Elke A Rundensteiner, and Matthew O Ward. 2009. Neighbor-based Pattern Detection for Windows over Streaming Data. In *Proceedings of the 2009 International Conference on Extending Database Technology*. 529–540.

[40] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2019. NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1303–1315.

[41] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2020. Ultrafast Local Outlier Detection from a Data Stream with Stationary Region Skipping. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1181–1191.

[42] Susik Yoon, Yooju Shin, Jae-Gil Lee, and Byung Suk Lee. 2020. The source code of MDUAL. https://github.com/kaist-dmlab/MDUAL. Accessed: 2021-02-26.

[43] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On Complexity and Optimization of Expensive Queries in Complex Event Processing. In *Proceedings of the 2014 ACM International Conference on Management of Data*. 217–228.

[44] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query Performance Prediction for Concurrent Queries using Graph Embedding. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1416–1428.