Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams

Kozue Ishida and Hiroyuki Kitagawa

[1] Graduate School of Systems and Information Engineering
[2] Center for Computational Sciences
University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305–8573, Japan
kozue-i@kde.cs.tsukuba.ac.jp, kitagawa@cs.tsukuba.ac.jp

Abstract. The development of sensor devices and ubiquitous computing have increased time-series data streams. With data streams, current data arrives continuously and must be monitored. This paper presents outlier detection over data streams by continuous monitoring. Outlier detection is an important data mining issue and discovers outliers, which have features that differ profoundly from other objects or values. Most existing outlier detection techniques, however, deal with static data, which is computationally expensive. Specifically, for outlier detection over data streams, real-time response is very important. Existing techniques for static data, however, are fraught with many meaningless processes over data streams, and the calculation cost is too high. This paper introduces a technique that provides effective outlier detection over data streams using differential processing, and confirms effectiveness.

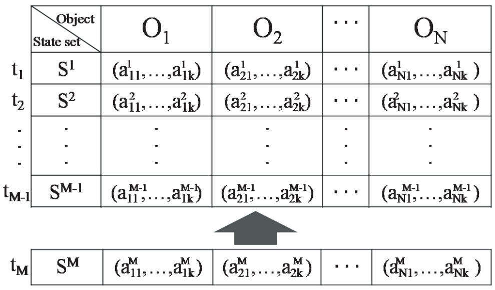
Key words: outlier detection, DB-Outlier, data stream, time-series data

1 Introduction

We face an explosive increase of data, so data mining, which identifies important information and knowledge, has become increasingly important. Outlier detection is a data mining issue; it discovers outliers with features that differ greatly from other objects or values. Applications include fraud detection, network intrusion detection, and financial analysis. Various outlier detection techniques over static data have been proposed, including the statistical-based approach [1, 2] and distance-based approach [3, 4].

The diffusion of sensor devices and improved ubiquitous computing have brought with them an increase in time-series data streams. Because data arrives continuously, the volume of data streams is very large. Data mining techniques for data streams are therefore important; the same applies to outlier detection.

Examples of outlier detection over data streams are: to monitor observation values from sensors continuously and detect outlier sensors that show values very different from other sensors, to monitor stock price movements of individual companies and detect outlier companies whose stock prices change very differently



	State set \ Object	O_1	O_2	···	O_N
t_1	S^1	$(a^1_{11},...,a^1_{1k})$	$(a^1_{21},...,a^1_{2k})$	···	$(a^1_{N1},...,a^1_{Nk})$
t_2	S^2	$(a^2_{11},...,a^2_{1k})$	$(a^2_{21},...,a^2_{2k})$	···	$(a^2_{N1},...,a^2_{Nk})$
⋮	⋮	⋮	⋮	⋮	⋮
t_{M-1}	S^{M-1}	$(a^{M-1}_{11},...,a^{M-1}_{1k})$	$(a^{M-1}_{21},...,a^{M-1}_{2k})$	···	$(a^{M-1}_{N1},...,a^{M-1}_{Nk})$
t_M	S^M	$(a^M_{11},...,a^M_{1k})$	$(a^M_{21},...,a^M_{2k})$	···	$(a^M_{N1},...,a^M_{Nk})$

Fig. 1. Arrival of a State Set S^M

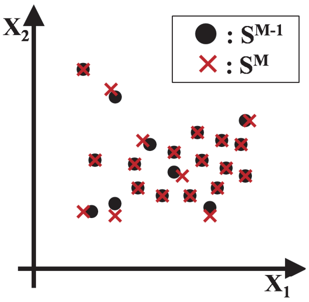


Fig. 2. Change of Data Distribution

from those of other companies, or to monitor the location of moving objects and detect outlier objects that are distantly-positioned from other objects.

When we have a set of objects that change state (such as observation values, internal states, or locations) over time, and if object O_i's state differs greatly from other objects' states at a given time, we can regard O_i as an outlier at the time. In continuously monitoring of a set of objects in which the state changes over time, we need to detect such outlier objects continuously.

Formally, when we have N objects whose state changes over time, let the set of objects' state at time t_j, $S^j = \{s^j_1, ..., s^j_N\}$, where each state tuple $s^j_i = (a^j_{i1}, ..., a^j_{ik})$ describes the state of object O_i at t_j. If the state of O_i, that is s^j_i, differs greatly from that of other objects, then we regard O_i as an outler at t_j.

We look at the problem of detecting outlier objects at current time t_M, where S^M arrives continuously as data streams $(S^1, ..., S^M)$, as shown in Fig. 1.

A straightforward solution to the problem is to apply the existing approach for static data, at every arrival of S^M. However, the yield of such an approach is too voluminous. Therefore, repeating the process for every time stamp is inefficient.

In general, a state set S^M is the change of the last state set S^{M-1}. In many cases, therefore, the differences between them are not great, as shown in Fig. 2. In cases like this, we can detect outliers effectively in S^M based on differential calculation using the result of outlier detection for S^{M-1}.

This paper proposes continuous outlier detection over data streams based on distance-based outlier detection (DB-Outlier), which is the basic approach to outlier detection. Experiments confirm effectiveness of the proposed approach. The rest of the paper is organized as follows: Section 2 mentions related work on outlier detection. Section 3 defines DB-Outliers and CDB-Outliers. Section 4 describes existing DB-Outlier detection methods as a preliminary to our proposal, and Section 5 explains continuous CDB-Outlier detection, our proposed method. Section 6 evaluates the effectiveness and efficiency of the algorithm compared with existing algorithms. Section 7 presents conclusions and future work.

2 Related Work

Various outlier detection methods have been proposed. This section introduces existing work on outlier detection over static data and stream data.

2.1 Outlier Detection over Static Data

The major methods of outlier detection over static data are as follows:

Statistical-based approaches [1, 2] assume that the dataset follows a statistical model. With these method we detect objects that deviate from the model as outliers. However, statistical approaches make a lot of assumptions (e.g., distribution model), and have difficulty dealing with high dimensional datasets.

Clustering approaches (e.g., CLARANS [5], DBSCAN [6], BIRCH [7], Wave Cluster [8], CLIQUE [9]) detect outliers as by-products. In most cases, the main objective is to find clusters in the dataset. For that reason, this method does not focus on outlier detection.

The density-based approach [10] adopts a Local Outlier Factor (LOF) which represents the local density of each data point's neighborhood and declares the degree of outlierness. The method detects objects having a high LOF as outliers.

The distance-based approach [3, 4] is a simpler and more common approach. It merely calculates the distance between two data points; distance is a common notion of many knowledge and technical areas. Effectiveness and importance are shown in [3, 4].

2.2 Outlier Detection over Stream Data

Because of increased interest, varied research is underway on data streams. The same is true for outlier detection. L. Su et al. propose outlier detection on distributed data streams [11] based on their original outlier model. K. Yamanishi et al. propose on-line outlier detection using statistical models [12, 13].

Compared with these approaches, our proposal is simpler and more versatile because we use the distance-based approach. Moreover, our proposed approach features efficiency based on differential processing.

3 Definition of Outliers

3.1 DB-Outlier

A DB-Outlier is defined by E. M. Knorr et al. as follows [4]:

Definition 1. An object O_i in a dataset S is a DB(p, D)-outlier if at least fraction p of the objects in S lie greater than distance D from O_i.

For an object O_i, the D-neighborhood of O_i contains the set of objects $O_q \in S$, where $d(O_i, O_q) \leq D$. Note that $d(O_i, O_q)$ denotes the distance between O_i and O_q. To simplify the discussion, we use another parameter $M[M = N(1-p)$, N : data size of $S]$, which is the maximum number of objects within the D-neighborhood of an outlier. k represents the dimension of S.

To clarify the definition, we show an example in Fig. 3.1. We have 30 objects (N=30) and parameter $p = 0.9$. Thus, if an object has at most $M[= 30(1 - 0.9) = 3]$ objects in its D-neighborhood, which is described as circles, then the object is a DB-Outlier. The left object has $2(\leq M)$ objects in its D-neighborhood, so it is a DB-Outlier. On the other hand, the right object is a non-outlier, because it has $12(> M)$ objects in its D-neighborhood.

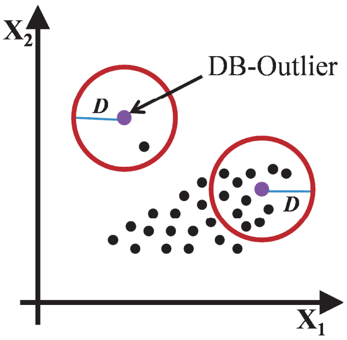


Fig. 3. DB-Outlier, $k = 2$, $p = 0.9$, $N = 30$

3.2 CDB-Outlier

In this paper, we call the current DB-Outlier in a data stream "CDB-Outlier." A data stream can be described as a state set $S^j(1 \leq j \leq M)$, which is illustrated in Fig. 1. We then provide the definition of CDB-Outlier in Definition 2.

Definition 2. An object O_i is a DB-Outlier at time t_j if tuple ${s_i}^j$ of O_i is a DB-Outlier in state set S^j. We call the DB-Outlier at current time t_M **CDB-Outlier**.

4 Algorithms for DB-Outlier Detection

This section presents DB-Outlier detection algorithms for static data [4]. We first explain the Simple algorithm. We then show the Cell-Based algorithm for the quick processing which is used in our proposal.

4.1 Simple Algorithm

This algorithm merely follows the definition of DB-Outlier. It applies the following processes for all objects in dataset S.

For each object O_i, we calculate the distance between O_i and other objects. Once there are more than M objects in the D-neighborhood, we stop the search and declare O_i as a non-outlier. Otherwise, we report O_i as an outlier.

The main drawback is time complexity $O(kN^2)$. This occurs because distance is calculated every 2 points until the object has been judged.

4.2 Cell-Based Algorithm

To skip distance calculations, the Cell-Based algorithm [4] employs a cell structure on a data space.

Cell Structure O_i's tuple $s_i^j = (a_{i1}^j, ..., a_{ik}^j)$ can be coded as a point in k-dimensional space with axes of X_1, ...,X_k. We divide this k-dimensional space into cells whose diagonal is $\frac{D}{2}$ length (length of the side $l = \frac{D}{2\sqrt{k}}$). Let $C_{x_1,...,x_k}$ describe the cell with x_1-th index of X_1 axis, ..., x_k-th index of X_k axis.

We then define L_1 neighbors and L_2 neighbors of $C_{x_1,...,x_k}$.

Definition 3. The L_1 neighbors of $C_{x_1,...,x_k}$, $L_1(C_{x_1,...,x_k})$ are the immediately neighboring cells of $C_{x_1,...,x_k}$, defined as follows,

$$L_1\ (C_{x_1,..,x_k}) = \{C_{u_1,...,u_k} \mid |u_i - x_i| \leq 1\ (1 \leq i \leq k) \wedge C_{u_1,...,u_k} \neq C_{x_1,...,x_k}\}.$$

Definition 4. The L_2 neighbors of $C_{x_1,...,x_k}$, $L_2(C_{x_1,...,x_k})$, are cells that satisfy the following formula,

$$L_2(C_{x_1,...,x_k}) = \{C_{u_1,...,u_k} \mid |u_i - x_i| \leq \lceil 2\sqrt{k} \rceil (1 \leq i \leq k) \wedge C_{u_1,...,u_k} \notin L_1(C_{x_1,...,x_k}) \wedge C_{u_1,...,u_k} \neq C_{x_1,...,x_k}\}.$$

Properties A.

(A1) If $O_i \in C_{x_1,...,x_k}$, $O_p \in C_{x_1,...,x_k}$, then $d(O_i, O_p) \leq \frac{D}{2}$.

(A2) If $O_i \in C_{x_1,...,x_k}$, $C_{u_1,...,u_k} \in L_1(C_{x_1,...,x_k})$, and $O_q \in C_{u_1,...,u_k}$, then $d(O_i, O_q) \leq D$.

(A3) If $O_i \in C_{x_1,...,x_k}$, $C_{u_1,...,u_k} \notin L_1(C_{x_1,...,x_k})$, $C_{u_1,...,u_k} \notin L_2(C_{x_1,...,x_k})$, $C_{u_1,...,u_k} \neq C_{x_1,...,x_k}$, and $O_r \in C_{u_1,...,u_k}$, then $d(O_i, O_r) > D$.

As shown in Fig. 4, it is obvious that property (A1) is met. Fig. 5 illustrates property (A2). $d(O_i, O_q)$ is max when the two objects are located as shown in this figure. Fig. 6 illustrates property (A3). $d(O_i, O_r)$ is minimum when the two objects are located as shown in this figure.

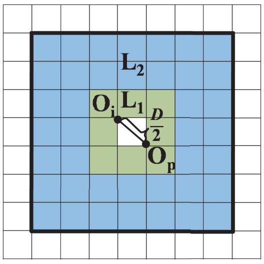


Fig. 4. Property (A1)

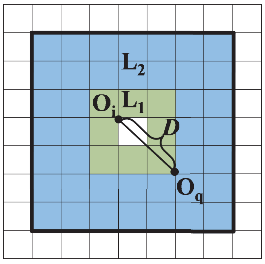


Fig. 5. Property (A2)

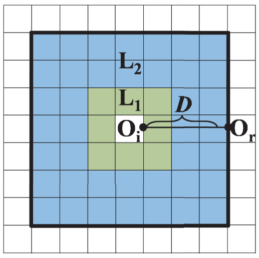


Fig. 6. Property (A3)

Let n, n_1, and n_2 be the numbers of objects in $C_{x_1,...,x_k}$, $L_1\ (C_{x_1,...,x_k})$, and $L_2\ (C_{x_1,...,x_k})$, respectively. Then we derive properties B from properties A as follows:

Properties B.

(B1) If $n > M$, $C_{x_1,\ldots,x_k}$ contains no DB-outliers.
(B2) If $n + n_1 > M$, $C_{x_1,\ldots,x_k}$ contains no DB-outliers.
(B3) If $n + n_1 + n_2 \leq M$, all objects in $C_{x_1,\ldots,x_k}$ are DB-outliers.

We color cells that satisfy (B1) as *red*, (B2) as *pink*, and (B3) as *yellow*. We can detect outliers in those cells without any distance calculation. Then we identify nonempty and uncolored cells as *white*. The decision for cell colors is called **CCD** (Cell-Color Decision). And the algorithm with CCD is as follows:

Algorithm Fig. 7 illustrates the Cell-Based algorithm. Step 2 quantizes each object to its appropriate cell. Step 3 labels all cells containing more than M objects, *red*(Property (B1)). Step 4 labels uncolored cells that have *red* cells in L_1 neighbors, *pink*(Property (B2)). Other cells satisfying Property (B2) are labeled *pink* in step 5b. Step 5cii labels cells satisfying Property (B3) as *yellow*, and reports all objects in the cells as DB-Outliers. Finally, uncolored cells (not satisfying Properties (B1), (B2), (B3)) are labeled as *white* in step 5ciii. CCD is a set of processes shown in step 3-5ciii1. We then operate only objects in *white* cells (C_ω) using an object-by-object process as follows: For each object $O_i \in C_\omega$, we calculate distance between O_i and each object O_q in cells $\in L_2(C_\omega)$, and count the number of objects in its D-neighborhood. We count $n + n_1$ in $Count_i$ in advance because all L_1 neighbors are always within the D-neighborhood. Once the number of objects in the D-neighborhood exceeds M, we declare O_i a non-outlier. If the count remains less than or equal to M after all calculation, we report O_i as an outlier. We call this process for each *white* cell (5ciii2) **WCP**(White-Cell Process), and the process for each object in a *white* cell (5ciii2a-c) **WOP**(White-Object Process).

Cell-by-cell basis decisions using Properties B help to determine whether or not an object in the cell is an outlier, and this leads to skipping a lot of distance calculations and reducing the process time compared with the Simple algorithm.

5 Proposed Method

This section describes continuous CDB-Outlier detection over data streams. That means we consider the detection of DB-Outliers when a state set S^M arrives, as shown in Fig. 1.

The straightforward approach to detecting CDB-Outliers is to process DB-Outlier detection for every state set S^M. However, in most cases, the data distribution of S^M is similar to that of S^{M-1}, so processing for all objects in S^M again has many meaningless calculations. Therefore, our proposed method does differential processing based on the change between S^{M-1} and S^M using the idea of the Cell-Based algorithm.

```
1. For each cell $C_q, n_q \leftarrow 0$
2. For each object $O_i$, do:
     Map $O_i$ to an appropriate cell $C_q$, store $O_i$, and increment $n_q$.
3. For each cell $C_q$, if $n_q > M$, label $C_q$ red.
4. For each red cell $C_r$, do:
     Label each of the $L_1$ neighbors of $C_r$ pink, provided the neighbor has not
     already been labeled red.
5. For each non-empty uncolored cell $C_w$, do:
   a. $n1_\omega \leftarrow \Sigma_{i \in L_1(C_\omega)} n_i$
   b. If $n_\omega + n1_\omega > M$, label $C_\omega$ pink.
   c. Else
     i. $n2_\omega \leftarrow \Sigma_{i \in L_2(C_\omega)} n_i$
     ii. If $n_\omega + n1_\omega + n2_\omega \leq M$, label $C_\omega$ yellow, mark all objects in $C_\omega$
        as outliers.
     iii. Else
       1. Label $C_\omega$ as white.
       2. For each object $O_i \in C_\omega$, do:
         a. $Count_i \leftarrow n_\omega + n1_\omega$
         b. For each object $O_q \in C_{L_2}, C_{L_2} \in L_2(C_\omega)$, if $(dist(O_i, O_q) \leq D)$:
            Increment $Count_i$ by 1. If $Count_i > M$, $O_i$ cannot be an outelir,
            so goto(5ciii2)
         c. Mark $O_i$ as an outlier
```

Fig. 7. Cell-Based Algorithm

5.1 Assumptions

We do differential processing with objects whose states changed from t_{M-1} to t_M (say **SC-Objects**(State-Change Objects)) and their current state sets $\Delta^M (= \{(O_i, s_i^M) \mid 1 \leq i \leq N \wedge s_i^{M-1} \neq s_i^M\})$. Our proposed method targets all state sets except the initial state set (S^1) of data streams. We use the original Cell-Based algorithm for S^1.

5.2 Differential Processing on an SC-Object

To simplify the problem, we consider the case with one SC-Object, O_P. There are two ways to change the state in the state space divided into cells. Let "$O_i \in C^j_{x_1,\ldots,x_k}$" represent that cell $C_{x_1,\ldots,x_k}$ contains O_i at t_j.

[**1**] Move to a different cell:
$O_P \in C^{M-1}_{x_1,\ldots,x_k}$, $O_P \in C^{M}_{x_1,\ldots,x_k}$,
$C^{M-1}_{x_1,\ldots,x_k} \neq C^{M}_{x_1,\ldots,x_k}$.

O_P influences cells at t_{M-1} and t_M and their L_1 and L_2 neighbors, which is described as colored area in Fig. 8. Since n of $C^{M-1}_{x_1,\ldots,x_k}$ and $C^{M}_{x_1,\ldots,x_k}$ change and influence those cells.

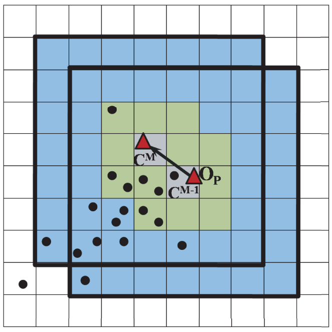


Fig. 8. Case[1]

[**2**] Move in the same cell:
$O_P \in C^{M-1}_{x_1,\ldots,x_k}$, $O_P \in C^{M}_{x_1,\ldots,x_k}$, $C^{M-1}_{x_1,\ldots,x_k} = C^{M}_{x_1,\ldots,x_k}$.

Because n of $C^{M}_{x_1,\ldots,x_k}$ does not change, *red*, *pink*, and *yellow* cells within L_2 neighbor are not influenced. It influences only *white* cells within the L_2 neighbor. There are two cases:

[**2a**] There are *white* cells in $L_2(C^{M}_{x_1,\ldots,x_k})$.
[**2b**] $C^{M}_{x_1,\ldots,x_k}$ itself is a *white* cell.

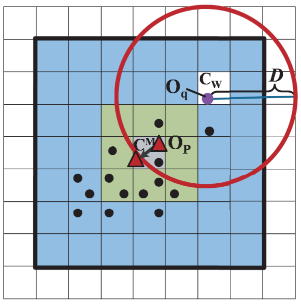


Fig. 9. Case[2a]

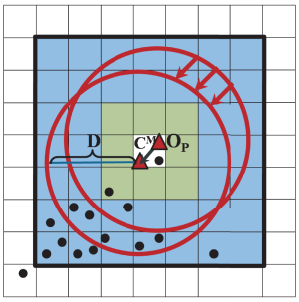


Fig. 10. Case[2b]

Fig. 9 illustrates the case [2a], and C_ω represents a *white* cell in $L_2(C^{M}_{x_1,\ldots,x_k})$. $O_q \in C_\omega$ can be influenced because O_P can enter or exit O_q's D-neighborhood. In case [2b], as shown in Fig. 10, the area of O_P's D-neighborhood will change, and O_P influences outlier decision of O_P.

5.3 Target Cells for Re-Outlier Detection

Actually, there are more than one SC-Objects from t_{M-1} to t_M. Therefore, we expand the above idea to more than one SC-Objects. We identify the target cells to reprocess, and classify them taking into account overlaps of each SC-Object's process. For example, the area of influence for case [1] contains that of case [2]. Targeted cells are the following 4 types.

Type A : Cells containing SC-Objects which have moved to or from another cell, at t_M (C^{M-1} and C^{M} in Fig. 8), namely
$TypeA = \{C_{x_1,\ldots,x_k} \mid (1 \leq i \leq N) \wedge ((O_i \in C^{M-1}_{x_1,\ldots,x_k} \wedge O_i \notin C^{M}_{x_1,\ldots,x_k}) \vee (O_i \notin C^{M-1}_{x_1,\ldots,x_k}) \wedge O_i \in C^{M}_{x_1,\ldots,x_k}))\}$.

Type B : Cells of L_1 and L_2 neighbors of Type A (Colored cells, in Fig. 8), except for those classified as Type A cells, namely
$TypeB = \{C_{x_1,\ldots,x_k} \mid (C_{u_1,\ldots,u_k} \in L_1(C_{x_1,\ldots,x_k}) \wedge C_{u_1,\ldots,u_k} \in TypeA) \vee (C_{u_1,\ldots,u_k} \in L_2(C_{x_1,\ldots,x_k}) \wedge C_{u_1,\ldots,u_k} \in TypeA) \wedge C_{x_1,\ldots,x_k} \notin TypeA\}$.

Type C : *White* cells that are in the L_2 neighbor of the cells that contain SC-Objects having moved within the same cell (C_ω in Fig. 9), except for those classified as Type A and B cells, namely
$TypeC = \{C_{x_1,\ldots,x_k} \mid C_{u_1,\ldots,u_k} \in L_2(C_{x_1,\ldots,x_k}) \wedge (1 \le i \le N) \wedge (O_i \in C^{M-1}_{u_1,\ldots,u_k} \wedge O_i \in C^{M}_{u_1,\ldots,u_k} \wedge C^{M-1}_{u_1,\ldots,u_k} = C^{M}_{u_1,\ldots,u_k} \wedge s_i^{M-1} \ne s_i^M) \wedge color(C_{x_1,\ldots,x_k}) = white \wedge C_{x_1,\ldots,x_k} \notin TypeA \cup TypeB\}$.

Type D : *White* cells that contain SC-Objects that have moved within the same cell (C^M in Fig. 10), except for those classified as Type A, B and C cells, namely
$TypeD = \{C_{x_1,\ldots,x_k} \mid (1 \le i \le N) \wedge (O_i \in C^{M-1}_{x_1,\ldots,x_k} \wedge O_i \in C^{M}_{x_1,\ldots,x_k} \wedge C^{M-1}_{x_1,\ldots,x_k} = C^{M}_{x_1,\ldots,x_k} \wedge s_i^{M-1} \ne s_i^M) \wedge color(C_{x_1,\ldots,x_k}) = white \wedge C_{x_1,\ldots,x_k} \notin TypeA \cup TypeB \cup TypeC\}$.

5.4 Algorithm

We only process the target cells with the proposed algorithm, shown in Fig. 11.

The input is a set of SC-Objects and its state at t_M, $\Delta^M (= \{(O_i, s_i^M) \mid (1 \le i \le N) \wedge s_i^{M-1} \ne s_i^M\})$. The output is CDB-Outliers.

Step 1 updates cells that contain SC-Objects at time t_M. If the cell C^M of O_P at t_M differs from C^{M-1} at t_{M-1}, then step 1a labels both cells, A(Case [**1**]). If there are *white* cells $C_\omega \in L_2(C^M)$, then step 1bi labels C_ω, C(Case [2a]). If the color of C^M is *white*, then step 1bii labels C^M, D(Case [2b]). Step 2 labels cells in L_1 and L_2 neighbor of Type A cells, B, and updates their n_1 and n_2. We have now targeted the cells to be reprocessed. We then classify the targeted cells based on labels A, B, C and D. Cells labeled A are in Type A, cells labeled B except for Type A cells are in Type B, cells labeled C except for Types A and B cells are in Type C, cells labeled D except for Types A, B, and C cells are in Type D. Step 3 does Re-CCD as shown in Fig. 12. Re-CCD differs from CCD because it does not count objects in cells of L_1 and L_2 neighbors, and uses only existing or updated n, n_1 and n_2. It also leads to reduced processing. Step 4 processes WCP, mentioned in the original Cell-Based algorithm (4.2), over Type C cells. Step 5 processes WOP over SC-Objects in Type D cells.The objects in *yellow* cells and outlier objects in *white* cells are output as CDB-Outliers.

6 Experiments and Results

Experiments with 2-dimensional (2-D) synthetic data and 3-dimensional (3-D) real data confirm improved processing time for the proposed method.

All of our tests were run on a Microsoft Windows Vista machine with an AMD Athlon(tm) 64 × 2 Dual Core Processor 3800+ 2GHz CPU and 1982MB of main memory. We implemented the software with Java 1.6.0_ 02.

This section explains the comparative approaches, describes the datasets, and provides the results and discussions.

```
Input: $\Delta^M$
Output: CDB-Outliers
    1. For each $(O_P, s_P^M) \in \Delta^M$, do: identify cell $C^M$ at $t_M$
        a. If($C^M \neq C^{M-1}$):
            i. Store $O_P$ in $C^M$, and update $n$ of $C^M$.
            ii. Delete $O_P$ from $C^{M-1}$, and update $n$ of $C^{M-1}$.
            iii. Label $C^M$ and $C^{M-1}$, A.
        b. Else:
            i. If there are white cells $(C_\omega)$ in $L_2(C^M)$, label $C_\omega$ C.
            ii. If color$(C^M) = white$, label $C^M$ D.
    2. For each cell $C_A$ of Type A, do:
        a. For each cell $C_{L1} \in L_1(C^M)$, do:
            i. Update $n_1$.
            ii. Label $C_{L1}$ B.
        b. For each cell $C_{L2} \in L_2(C^M)$, do:
            i. Update $n_2$.
            ii. Label $C_{L2}$ B.
    3. For each cell $C_{AB}$ of Type A or B, do: Re-CCD.
    4. For each cell $C_C$ of Type C, do: WCP.
    5. For each cell $C_D$ of Type D, do:
        a. For each SC-Object $O_P \in C_D$, do: WOP.
```

Fig. 11. Proposed Algorithm ($t_{M-1} \rightarrow t_M$)

```
1. If $n > M$, color red.
2. Else if $n + n_1 > M$, color pink.
3. Else if $n + n_1 + n_2 \leq M$, color yellow.
4. Else, do: WCP, color white.
```

Fig. 12. Re-CCD

6.1 Comparative Approaches

We compared our proposed method with two DB-Outlier methods.

CM(Cell-Based Method): A method that executes the Cell-Based algorithm for every time stamp.
SM(Simple Method): A method that executes the minimal process of the Simple algorithm with Δ^M. It processes only SC-Objects and the neighbor objects that can be influenced by SC-Objects.

Applying the original Simple algorithm for every time stamp takes huge computation time and is beyond comparison. Hence, we do not show the comparison.

6.2 Datasets

MO Data (2-D) We use moving objects' data streams (MO data) generated by the Mobi-REAL[3] simulator. Fig. 13 describes the distribution of moving objects

[3] http://www.mobireal.net/index.html

at a given time, and each point represents each moving object. We set passes for moving objects: dense at the skirt area and sparse at the central area. Therefore, objects passing the central area are detected as CDB-Outliers. We use the x and y coordinates for each object per time unit (1sec) as 2-D datasets. The area of this simulation is 700 × 700[m^2]. Parameters are: the number of objects is 10000, the (SC-Objects)/(all objects) per time unit is 50[%], the average of moved distance of the SC-Objects per time unit is 1.5[m], $D = 15$[m], and $p = 0.9995$.

Stock Data (3-D) We use daily stock dataset (Dec., 2007, Japan) [14] as a 3-D real dataset. We detect brands that behave very differently from others. This dataset consists of the band, closing stock price, and completed amount and changes per day. To reduce the non-uniformity of each attribute, we assign weights as band $\times 10^3$, (closing stock price) $\times 10^2$. Fig. 14 shows data distribution at a given time. The data has 4039 objects. The average of (SC-Objects)/(all objects) per time unit is 83% .

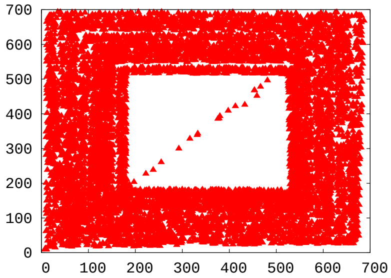


Fig. 13. MO Data

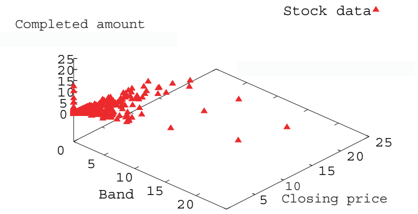


Fig. 14. Stock Data ($\times 10^5$)

6.3 Results and Discussions

We compared the processing time for the proposed method (PM) and comparative method (CM, SM), and evaluated the relationship with each parameter.

MO data (2-D) Figs. 15-20 shows results of MO data. In 2-D cases, SM takes too long (around 30000(ms)) to process a dataset per 1000(ms), so we can say that SM is not appropriate at all. Thus, we evaluate only CM and PM in 2-D experiments. Fig. 15 shows time versus the ratio of SC-Objects in all objects per second. Even if the ratio of SC-Objects reaches 100%, PM takes less time than CM, about 70% of CM. This occurs because, even if all objects change in state, not all objects move to different cells. Further, CM counts objects in L_1 and L_2 neighbors every time. Fig. 16 shows the relationship between time and the number of all objects. Both methods take more time as the number increases. PM exceeds CM in all cases. Fig. 17 shows time versus the average of moved distance of SC-Objects. With this result, we cannot see clear relation to each other. Fig. 18 illustrates the correlation between time and p. CM and PM shorten time as p grows. When p is large, the difference between CM and PM becomes large. In Fig. 19, we change D. At around $D = 30$, PM time approximately equals to CM. This occurs because, as shown in Fig. 20, the sum of reprocessed cells in PM is almost equal to the sum of all cells around $D = 30$. Actually, there are no CDB-Outliers for $D > 25$.

From the results, in many cases, PM is effective for continuous processing in 2-D datasets. However, if reprocessed cells in PM increase, PM loses its advantage.

Stock Data (3-D) Fig. 21 shows the relationship between time and D, where $p = 0.997$. It is obvious that CM is profoundly affected by D, and if we have small D, the process time is very long. With 3-D data, the number of all cells is bigger than that of 2-D data. Moreover, the number of cells in L_1 and L_2 neighbors is also bigger. Therefore the calculation cost is higher. On the other hand, PM is little influenced by D, since we use differential calculation. Fig. 22 is the relationship between process time and p, where $D = 200 \times 10^5$. SM is profoundly affected by p, because the smaller the p, the bigger the M. Therefore, distance calculations until "the object is not an outlier" increase. On the other hand, PM and CM are not significantly influenced because of cell-by-cell decisions.

PM can maintain the advantage of the cell-by-cell process with 3-D datasets. Therefore, PM takes shorter than CM and SM. With those results, we can say PM is also effective for 3-D datasets.

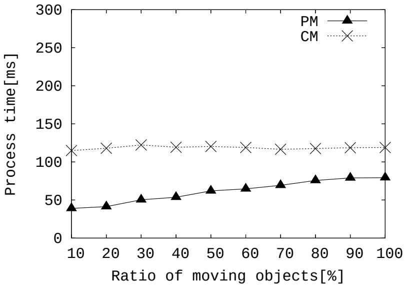


Fig. 15. MO Data: Time versus SC-Objects/All Objects

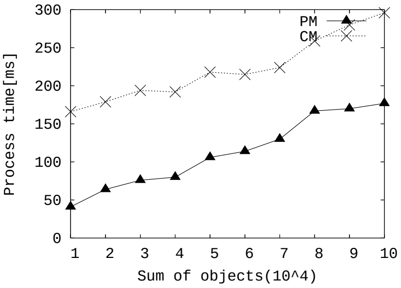


Fig. 16. MO Data: Time versus the Number of All Objects

7 Conclusions and Future Work

In this paper, we have proposed continuous outlier detection over data streams. We employ DB-Outlier, and based on the Cell-Based algorithm for quick processing, we provide an effective algorithm using differential processing over data

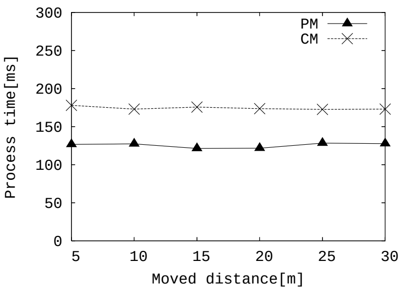


Fig. 17. MO Data: Time versus the Average Moved Distance

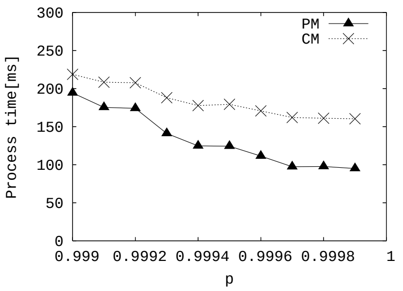


Fig. 18. MO Data: Time versus p

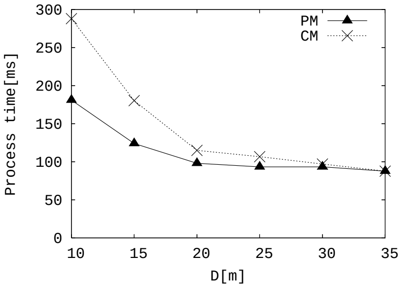


Fig. 19. MO Data: Time versus D

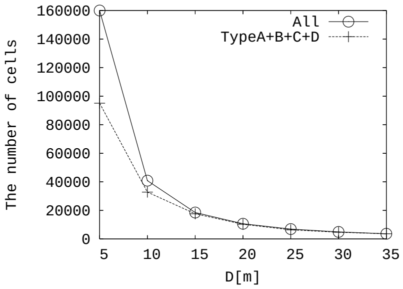


Fig. 20. MO Data: Number of Reprocessed Cells versus D

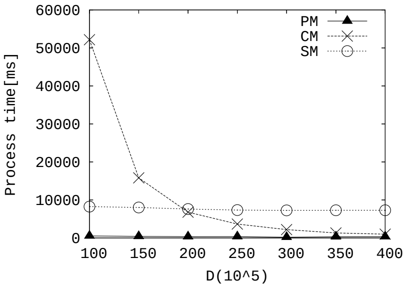


Fig. 21. Stock data: Time versus D

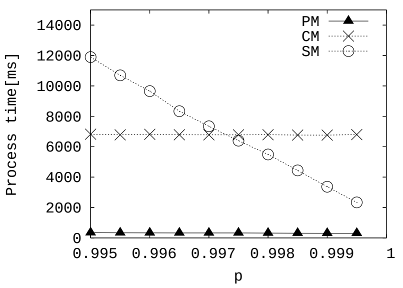


Fig. 22. Stock Data: Time versus p

streams. We evaluated our proposed method with synthetic and real datasets, and showed its advantage over naive methods. Extensions to cope with high dimensional data and dynamic data streams are interesting future research issues.

Acknowledgment. This research has been supported in part by the Grant-in-Aid for Scientific Research from MEXT (# 19024006).

References

1. Barret, V., Lewis, T.: Outliers in Statistical Data. Wiley, Chichester, (2001)
2. Eskin, E.: Anomaly Detection over Noisy Data using Learned Probability Distributions. ICML, pp. 255–262. (2000)
3. Knorr, E.M., Ng, R.T.: Finding Intensional Knowledge of Distance-Based Outliers. VLDB, pp. 211–222. (1999)
4. Knorr, E.M., Ng, R.T., Tucakov, V.: Distance-Based Outliers: Algorithms and Applications. VLDB J. 8(3-4), pp. 237–253. (2000)
5. Ng, R.T., Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining. VLDB, pp. 144–155. (1994)
6. Ester, M., Kriegel, H.P., Xu, X.: A Database Interface for Clustering in Large Spatial Databases. KDD, pp. 94–99. (1995)
7. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. SIGMOD, pp. 103–114. (1996)
8. Sheikholeslami, G., Chatterjee, S., Zhang, A.: WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. VLDB, pp. 428–439. (1998)
9. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. SIGMOD, pp. 94–105. (1998)
10. Breunig, M.M., Kriegel,H.P., Ng, R.T., Sander, J.: LOF: Identifying Density-Based Local Outliers. SIGMOD, pp. 93–104. (2000)
11. Su, L., Han, W., Yang, S., Zou, P., Jia, Y.: Continuous Adaptive Outlier Detection on Distributed Data Streams. HPCC, pp. 74–85. (2007)
12. Yamanishi, K., Takeuchi, J., Williams, G., Milne, P.: On-line Unsupervised Outlier Detection using Finite Mixtures with Discounting Learning Algorithms. KDD, pp. 320–324. (2000)
13. Yamanishi, K., Takeuchi, J.: Discovering Outlier Filtering Rules from Unlabeled Data: Combining a Supervised Learner with an Unsupervised Learner. KDD, pp. 389–394. (2001)
14. ITicker, `http://homepage1.nifty.com/hdatelier/`