LODC: A Lightweight Online Update Method for Density-Based Clustering

Jie Jiang, Junhua Fang(✉), Pingfu Chao, Pengpeng Zhao, and An Liu

Department of Computer Science and Technology, Soochow University, Suzhou, China
20235227074@stu.suda.edu.cn, jhfang@suda.edu.cn, pfchao@suda.edu.cn, ppzhao@suda.edu.cn, anliu@suda.edu.cn

Abstract. Density-based clustering methods, known for their capability in detecting clusters with arbitrary geometries, consistently attract significant research attention for enhancing their computational efficiency and accuracy. Most existing research focuses on batch processing of static datasets. In recent years, real-time clustering computation has become increasingly critical in various scenarios, such as traffic planning and environmental monitoring. However, the incremental computation method for real-time processing faces the following two issues: (1) The computational accuracy cannot be guaranteed because tracking cascading cluster transformations is difficult. (2) The computational efficiency is hindered due to the uncertainty in range searches during updates, which substantially increases processing time. To overcome these issues, this paper proposes a novel Lightweight Online update method for Density-based Clustering (LODC). Specifically, LODC optimizes density-reachability and clustering-related concepts to streamline the tracking of clustering evolution. Additionally, LODC employs an evaluation model to minimize the regions involved in range searches, thereby providing a lightweight computational approach. Extensive experiments demonstrate that LODC outperforms existing methods regarding update latency and achieves robust clustering quality.

Keywords: Density-based clustering · Incremental computation · DBSCAN.

1 Introduction

Background. With the ubiquity of wireless networks and the widespread adoption of GPS devices, large volumes of spatiotemporal data capturing object movements can now be efficiently collected [21]. The utilization of these datasets for cluster analysis not only identifies patterns in the motion of mobile objects but also enables kinetic feature extraction and knowledge management. Recently, there has been an escalating demand for real-time monitoring applications across various domains. For instance, continuous traffic flow surveillance [13], the evolutionary analysis of crowd clustering [18], and tracking wildlife migration patterns [1]. These scenarios present new challenges for clustering-based applications, specifically the ability to dynamically and real-time output clustering results.

Motivations. Among various clustering algorithms, density-based methods have garnered considerable attention due to their ability to identify clusters of arbitrary shapes [11]. However, these methods face computational challenges, as they require exhaustive traversal of objects to obtain density information, resulting in a time complexity of $O(n^2)$, where n denotes the number of objects. This high complexity renders traditional density-based approaches unsuitable for incremental computations in real-time scenarios. These limitations motivate us to address two fundamental objectives: (1) Design specialized data structures and operational workflows for different tasks to ensure the correctness of computational results. (2) Minimize unstructured range searches during computations by redefining the computational regions to better suit real-time scenarios.

Challenges. To avoid unstructured searches during clustering computations while maintaining result accuracy, the design of incremental clustering methods for real-time scenarios inevitably faces the following two challenges: (1) Computational accuracy: In real-time scenarios, newly arriving points and expired points are processed through insertion and deletion operations, respectively. These operations can have cascading effects on the overall clustering state. The update of a single point impacts the density-reachability of neighboring points and contributes to the evolution of cluster structures. Six distinct patterns of evolution can be identified: emergence, expansion, and merge for insertions, as well as dissipation, reduction, and split for deletions. (2) Computational efficiency: To mitigate the performance degradation caused by global search operations, localized computation is often the preferred approach. However, ensuring the accuracy of results during localized computation poses a significant challenge. Specifically, it is crucial to define appropriate data structures and roles to quickly determine the range of cluster evolution affected by the update of a point.

Contributions. In response to the aforementioned challenges, we propose a density-based clustering algorithm, named LODC, which is suitable for incremental computation in real-time scenarios. By leveraging density-reachability between adjacent cells, LODC assigns novel roles to cells to reduce the computational overhead required during updates, thereby addressing the challenge of computational efficiency. Additionally, LODC quickly identifies the type of evolution and iteratively updates cluster labels using cell-BFS (Breadth-First Search), effectively tackling the challenge of ensuring the accuracy of clustering results. The main contributions of this work are summarized as follows:

- We propose a clustering computation framework tailored for incremental computation scenarios. Through the design of data initialization mapping and specific insertion and deletion operations, the framework ensures a lightweight computation process for incremental clustering.
- We design specialized data structures for incremental clustering operations. Additionally, by defining the roles of computational objects and devising specific algorithmic procedures, the results correctness can be guaranteed when handling real-time data insertion or deletion.

- Extensive experiments under various parameter settings demonstrate that LODC significantly outperforms comparative methods regarding update latency and maintains high clustering quality during continuous updates.

2 Related Work

Building upon the foundational DBSCAN framework [4], numerous density-based clustering variants have been developed. Depending on the application scenarios, research on density-based clustering can be broadly categorized into studies on static DBSCAN and dynamic DBSCAN.

Static density-based clustering. RP-DBSCAN [15] is a parallel version of DBSCAN designed to address load imbalance and inefficiency in skewed datasets. DBSCAN++ [10] is an enhancement of the original DBSCAN algorithm that significantly accelerates processing for large-scale static datasets. Grid-based DBSCAN [2] employs a Hierarchical Grid-Based (HGB) structure to enhance the efficiency of neighboring grid queries, which includes an index for non-empty grids. GriT-DBSCAN [8] proposes a novel grid tree structure that facilitates efficient queries of neighboring grids, achieving linear time complexity.

Dynamic density-based clustering. IncDBSCAN [5] is the first algorithm achieving incremental updates in density-based clustering. Extra-N [19] capitalizes on the predictability of data points in the future. Gan and Tao [6] propose an approximate DBSCAN algorithm that is applicable in both static and incremental scenarios based on grid partition. StreamSW [14] uses p-micro-cluster and grids [16] to temporarily store information from data streams.

However, static algorithms are heavily limited in processing real-time data because they require recalculating clustering results with each update. In contrast, dynamic algorithms suffer from limited scalability in various application scenarios. For instance, the effectiveness of DenForest [12] diminishes in scenarios characterized by frequent insertions, while DBSTREAM [7] lacks deletion support. To overcome this, we propose a novel incremental density-based clustering algorithm that can update cluster formations in real-time processing scenarios and achieve insertion and deletion updates.

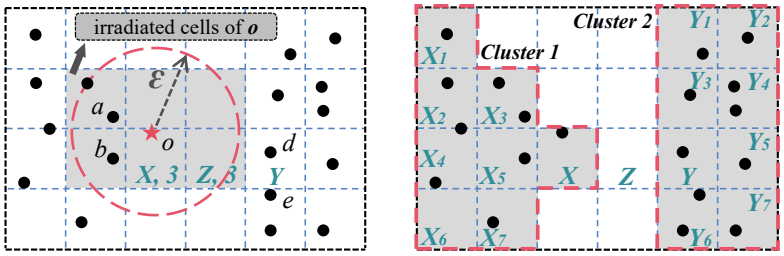
3 Preliminaries

3.1 Basic Concepts

The DBSCAN clustering algorithm defines two essential parameters: the distance threshold ϵ and the density threshold $MinPts$ [4]. Similarly, these parameters are also used in LODC. The data space $\mathbb{R}$ is uniformly divided into a regular grid structure, with each cell having a side length of d. In this paper, unless expressly stated otherwise, the default value of the side length d is set to $\frac{\sqrt{2}}{2}\epsilon$, ensuring that all points in the same cell are within a distance of ϵ of each other. Building upon this grid structure, we develop key definitions and assign functional roles to cells, enabling the tracking of dynamic changes in clusters.

Definition 1 (Radiation range & irradiated cell). *The **radiation range** of point p is defined as the aggregate of cells encompassed by a circle centered at p and extending a radius of ϵ. A cell is an **irradiated cell** of p if its geometric center lies inside the circle's boundary.*

Figure 1(a) shows the radiation range of point o. The shaded area in the circle centered at o with a radius of ϵ represents the irradiated cells of o, as the centers of these cells lie within the circle. Conversely, cells beyond this delineated boundary, such as Y, are not considered irradiated cells of o.



(a) Radiation range (b) Density reachable

Fig. 1. Illustration of fundamental concepts in LODC. The $Minpts = 3$, $d = \frac{\sqrt{2}}{2}\epsilon$. X, Y, Z, X_i, and Y_i represent cells, while a to e represent points.

Definition 2 (Cell-count). *The **cell-count** of a cell is a counter which records the number of points whose radiation range covers that cell.*

The cell-count of X is 3 because X is within the radiation range of a, b, and o in Figure 1(a). Upon removal of o from the dataset, the cell-count of X is synchronously updated to 2. Similarly, the cell-count for all cells within the radiation range of o must be updated. Core cells in LODC serve as the fundamental basis for cluster formation and facilitate connections among various parts to establish clusters.

Definition 3 (Core cell). *Cell X is a **core cell** if its cell-count meets the density threshold $MinPts$ and contains at least one point.*

In Figure 1(a), cell X qualifies as a core cell, whereas cell Z does not. This is because the cell-count of X meets the density threshold $MinPts$ of 3 and contains point o. Although Z lies within the radiation range of points o, d, and e, it contains no points and therefore cannot be classified as a core cell.

Definition 4 (Adjacent cell). *Cell Y is an **adjacent cell** of X if Y is directly contiguous to X, including cells that share either an edge or a corner with X.*

Cell Z in Figure 1(a) is an adjacent cell of X. However, cell Y is not an adjacent cell of X since Y does not share an edge or a corner with X. In two-dimensional space, this set typically includes up to eight adjacent cells.

Definition 5 (Border cell & noise cell). *Cell X is a **border cell** if it is not a core cell but is adjacent to at least one core cell, irrespective of whether any data points fall within X. Otherwise, it is considered a **noise cell**.*

As illustrated in Figure 1(a), Z is classified as a border cell. Although cell Z currently contains no points, it is an adjacent cell of the core cell X (or cell Y).

In density-based clustering algorithms, density-reachability is crucial for cluster formation. We have modified the concepts of density-reachability to accommodate cell-based processing, which differ significantly from DBSCAN.

Definition 6 (Cell directly density-reachable). *Cell Y is **cell directly density-reachable** from core cell X if Y is an adjacent cell of X.*

Definition 7 (Cell density-reachable). *Cell X is **cell density-reachable** from Z if there is a sequence of cells Y_1, ..., Y_n, $Y_1 = Z$, $Y_n = X$, where each Y_{i+1} is cell directly density-reachable from Y_i.*

Definition 8 (Cell density-connected). *Cell X and Y are said to be **cell density-connected** if they are cell density-reachable from the same core cell O.*

Definition 9 (Clusters in LODC). *A cluster C in LODC is a non-empty subset satisfying the following conditions:*
1) $\forall X \in C$: If cell Y is cell density-reachable from X, then $Y \in C$.
2) $\forall X, Y \in C$: Y is cell density-connected to X.

LODC considers only adjacent cells to define density-reachability rather than a distance threshold. Figure 1(b) shows a typical example that contains two clusters, named *Cluster 1* and *Cluster 2*. The cell X is directly adjacent to X_3, X_5 and X_7, making them cell directly density-reachable from X. Additionally, core cell X_4 is cell density-reachable from X via core cells $X_4 \rightarrow X_5 \rightarrow X$. Cell X_1 and X_7 are cell density-connected since they are cell density-reachable from X. Additionally, each core cell or border cell acquires a unique identifier, ***cid***, indicating which cluster the cell belongs to and distinguishing it from cells of other clusters. Border cells inherit the *cid* of adjacent core cells, while data points inherit the *cid* of their host cells. For instance, all cells in *Cluster 1* in Figure 1(b) are assigned a unique cluster identifier, such as *cid* 1, to indicate their membership in a specific cluster. All points in *Cluster 1* share the same *cid* as their corresponding cell, which is also 1.

3.2 Problem Statement

The goal of this paper is to develop a fully dynamic incremental clustering algorithm in real-time scenarios. Based on the above definitions, the problem can be stated as: Given a continuously evolving two-dimensional dataset $\mathcal{D} =$

$\{d_1, d_2, \cdots\}$ where d_i is the *i-th* data instance, $\mathcal{D}$ evolves as new points are added ($\Delta\mathcal{D}_{in}$) and old points are removed ($\Delta\mathcal{D}_{out}$). LODC analyzes the impact of these updates on existing clusters, promptly updating clustering results and identifying the type of cluster evolution. In other words, implementing LODC is generally formalized as discovering and continually updating clusters effectively as $\mathcal{D}$ evolves.

4 LODC Framework

As depicted in Figure 2, the overall clustering process is composed of two key components. The first component is responsible for collecting the cells that require updating. The second is designed to update the clustering structure. The details are outlined as follows:

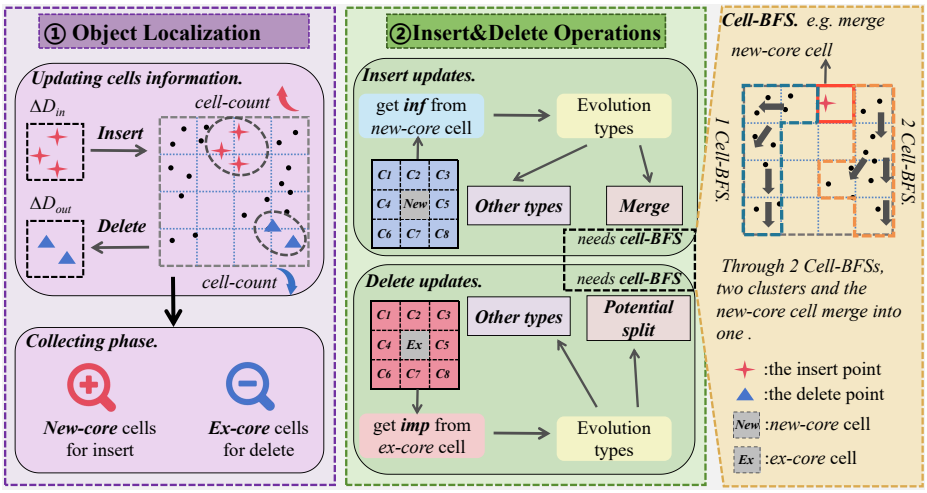


Fig. 2. Framework of LODC. The clustering process of LODC consists of two parts. ① Input: the set of updated points $\Delta\mathcal{D}$. Output: the set of execution units *new/ex-core* cells for insert/delete operations. ② Input: each *new/ex-core* cell. Output: the updated cluster structure.

1) **Object localization.** When dealing with the real-time updated data $\Delta\mathcal{D}$, LODC first maps these points to cells and updates cell information. Then, LODC collects *new-core* cells for insert operations and *ex-core* cells for delete operations (Detailed in Section 4.1).

2) **Insert and delete operations.** For each *new/ex-core* cell, LODC retrieves *inf/imp* for insertions and deletions to determine the evolution types. Subsequently, LODC executes cell-BFS depending on evolution types, completing the clustering updates. The cell-BFS can traverse the entire cluster to which the starting cell belongs (Detailed in Sections 4.2 and 4.3).

4.1 Object Localization

The update procedure inevitably alters cell density-reachability states, leading to changes in cluster structures. Specifically, insertion may convert non-core cells into core cells, while deletion may cause original core cells to lose their core status. The potential evolution types can be classified into six distinct categories: emergence, expansion, merge, dissipation, reduction, and split. In this paper, we define the cells affected by these updates as *new-core* cells and *ex-core* cells, respectively.

Definition 10 (*New-core* cell & *ex-core* cell). *Let $\mathcal{X}$ represent the current set of all cells, and $\mathcal{X}^+$ and $\mathcal{X}^-$, respectively, represent the states of cells after the insertion and deletion of p:*

*1) Z is a **new-core** cell if Z is not a core in $\mathcal{X}$, but becomes a core in $\mathcal{X}^+$.*

*2) Z is an **ex-core** cell if Z is a core in $\mathcal{X}$, but is no longer a core in $\mathcal{X}^-$.*

Algorithm 1: Collect-NEW

Input : $\mathcal{X}$, ϵ, $MinPts$, point p
Output: $NewCoreCell$
function Collect-NEW$(\mathcal{X}, p, \epsilon, MinPts)$**:**
  $NewCoreCell \leftarrow \emptyset$;
  for *each cell in p's irradiated cells* **do**
    $cell.cell\text{-}count++$;
    if *cell is a* ***new-core*** **then**
      $NewCoreCell \leftarrow cell$;
      $cell.label \leftarrow core$;
  return $NewCoreCell$;

Algorithm 2: Collect-EX

Input : $\mathcal{X}$, ϵ, $MinPts$, point q
Output: $ExCoreCell$
function Collect-EX$(\mathcal{X}, q, \epsilon, MinPts)$**:**
  $ExCoreCell \leftarrow \emptyset$;
  for *each cell in q's irradiated cells* **do**
    $cell.cell\text{-}count--$;
    if *cell is an* ***ex-core*** **then**
      $ExCoreCell \leftarrow cell$;
      $cell.label \leftarrow uncore$;
  return $ExCoreCell$;

If an update operation generates neither *new-core* nor *ex-core* cells, this indicates that it does not impact the structure of the cluster. Conversely, it necessitates a comprehensive analysis of the type of cluster evolution. Algorithm 1 and Algorithm 2 provide the pseudocodes for Collect-NEW and Collect-EX, respectively. When a point p is inserted, *new-core* cells may be created, thereby establishing new cell density-reachability. Collect-NEW gathers the *new-core* cells among the irradiated cells of p. Similarly, when a point q is deleted, *ex-core* cells may be created, disrupting the existing cell density-reachability. Collect-EX extracts the *ex-core* cells from the irradiated cells of q. As illustrated in Section 3, cell Z is a border cell in Figure 1(b). If a point p is inserted to cell Z, Collect-NEW updates the cell-count for the irradiated cells associated with p, gathering *new-core* cells. Given that the cell-count of Z already meets the $MinPts$ and contains point p now. Consequently, inserting p causes Z to become a *new-core* cell.

4.2 Insert Operation

To complete insertion, LODC uses *inf* to record the adjacent cells' clustering information. The value of *inf* represents the number of distinct clusters in the adjacent cells of a *new-core* cell. Based on the three types of evolution caused by insertions, we categorize *inf* into three categories. Given the *inf* of a *new-core* cell, the cluster evolution can be classified as:

1) $\boldsymbol{Inf = 0}$. This indicates the absence of other cores in its adjacent cells, corresponding to the **emergence** of a new cluster.

2) $\boldsymbol{Inf = 1}$. This signifies the presence of an existing cluster in its adjacent cells. The cluster will assimilate the *new-core* cell, leading to an increase in size, which signifies its **expansion**.

3) $\boldsymbol{Inf > 1}$. This suggests the presence of multiple clusters in its adjacent cells. The *new-core* cell will establish density-reachability relationships with neighboring clusters, merging them into a single unified cluster. The evolution type is **merge**.

As depicted in Figure 2, cell *New* represents a *new-core* cell. If none of its eight adjacent cells contain a core, the value of *inf* is 0. However, if both C_1 and C_2 are cores, then, based on cell density-reachability, they belong to the same cluster, resulting in an *inf* value of 1. The theoretical maximum value of *inf* is 4, which occurs only when C_1, C_3, C_6, and C_8 are cores and belong to different clusters. In this case, the merging of four distinct clusters takes place.

Algorithm 3 illustrates the pseudocode for INSERT. Given the inserted point p, Collect-NEW is executed to obtain *NewCoreCell*, which serves as the basis for the insertion updates. The traversal is then performed through *NewCoreCell*. Here, the *point to point* BFS has changed into *cell to cell* BFS. By simply providing a starting cell index Z within a cluster, cell-BFS can quickly retrieve information from adjacent cells and traverse the entire cluster to which the initial cell belongs. For each *new-core* cell, the *inf* value is calculated based on its adjacent cells (Lines 4~10).

1) $Inf = 0$: **emergence**, assign a new *cid* to the new core cell.

2) $Inf = 1$: **expansion**, unify the *cid* of the *new-core* cell with the nearby cluster, ensuring that the *new-core* cell is integrated into the existing cluster.

3) $Inf > 1$: **merge**, traverse all clusters using cell-BFS and synchronize the *cid* values of all cells. The maximum value of *inf* is 4, indicating that up to four clusters can merge, which can be accomplished with up to four cell-BFS operations.

Example 1: Figure 3(a)~3(c) illustrate a process of insert operation with $MinPts = 1$, where any cell containing points is considered a core cell. Initially, Figure 3(a) shows two clusters, *Cluster 1* and *Cluster 2*, formed by eleven points. When a new point p is inserted into cell Z, LODC executes the Collect-NEW operation to update the irradiated cells and identifies Z as a new core cell ($NewCoreCell = \{Z\}$). The INSERT function then processes Z. In Figure 3(b), it is observed that the adjacent cells belong to two distinct clusters: $\{X, X_4\}$ from *Cluster 1* and $\{Y, Y_1, Y_4\}$ from *Cluster 2*, yielding an *inf* value of 2. The *inf* value indicates the need to merge these clusters. The merging process begins

Algorithm 3: INSERT

```
   Input  : 𝒳, NewCoreCell
   Output: 𝒳′ (𝒳 after update)
1  function
     INSERT(𝒳, NewCoreCell):
2    if NewCoreCell ≠ ∅ then
3      for each cell in
         NewCoreCell do
4        inf ← adjacent cells;
5        if inf = 0 then
6          Emergence;
7        else if inf = 1 then
8          Expansion;
9        else
10         Merge;
11     Update noise cells;
12   else
13     The cluster structure
         remains unchanged;
14   return 𝒳′;
```

Algorithm 4: DELETE

```
   Input  : 𝒳, ExCoreCell
   Output: 𝒳′ (𝒳 after update)
1  function
     DELETE(𝒳, ExCoreCell):
2    if ExCoreCell ≠ ∅ then
3      for each cell in
         ExCoreCell do
4        imp ← adjacent cells;
5        if imp = 0 then
6          Dissipation;
7        else if imp = 1 then
8          Reduction;
9        else
10         Potential split;
11     Update border cells;
12   else
13     The cluster structure
         remains unchanged;
14   return 𝒳′;
```

by assigning a new *cid* to Z, followed by two cell-BFSs traversal from any cell within each cluster to update the *cid* of all traversed cells. For instance, cell-BFS starting from X updates the *cid* values of cells X_4, X_3, X_2 and X_1. Similarly, the process is repeated starting from Y. This procedure merges *Cluster 1* and *Cluster 2* into a new cluster, namely, the *Cluster 3* illustrated in Figure 3(c).

Time Complexity Analysis. The time overhead of the insertion operation consists of two parts: Collect-NEW and INSERT. In Collect-NEW, most of the time is spent checking the cells within the radiation range, which involves a minimal number of cells. Suppose this part needs to check k cells, then the time overhead is $O(k)$. In INSERT, the value of *inf* can be obtained with only eight accesses, and the time spent on obtaining *inf* can be considered negligible. If *inf* is 0 or 1, the subsequent update operations can be completed within amortized $O(1)$ time. The longest time overhead occurs in the case of cluster merging, where different clusters need to be traversed (the value of *inf* does not exceed 4). Suppose each cluster has an average of $\bar{m}$ cells, then the time overhead of this step is $O(inf \cdot \bar{m})$. Assuming the probability of cluster merging is P_m, then the total time overhead of the INSERT operation is: $O(k) + P_m \cdot O(\inf \cdot \bar{m}) + (1 - P_m) \cdot O(1)$. Since k and *inf* are small constants, the time complexity of the insert operation is $O(\bar{m})$.

4.3 Delete Operation

LODC uses *imp* to record the clustering information of adjacent cells during deletion operations. Prior to the update, *ex-core* cells and the cores within their adjacent cells belong to the same cluster. The *imp* represents the number of cell

density-reachability connections locally present among the eight adjacent cells of an *ex-core* cell. Connections beyond these adjacent cells are not considered. For example, cell *Ex* represents an *ex-core* cell, as shown in Figure 2. If C_1, C_2, and C_5 are core cells, they form a local cell density-reachability relationship, resulting in an *imp* value of 1. Similarly, if C_1, C_2, and C_7 are core cells, C_1 and C_2 form one local cell density-reachability relationship, while C_7 remains independent, yielding an *imp* value of 2. Additionally, similar to *inf*, the maximum possible value of *imp* is 4.

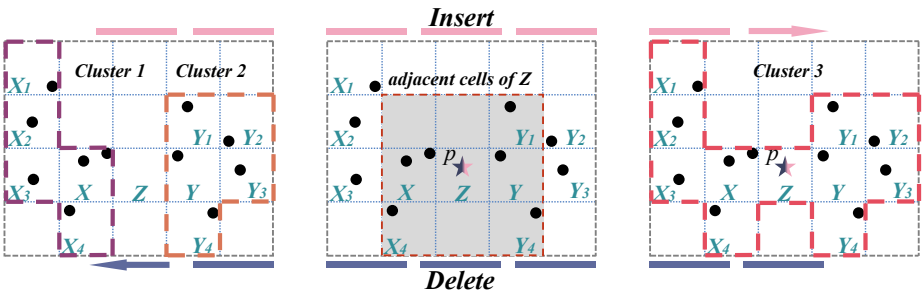
Notably, LODC uses **potential split** instead of **split**. Based on the three types of evolution caused by deletions, we divide *imp* into three categories. Given the *imp* of an *ex-core* cell, the cluster evolution can be classified as:

1) ***Imp* = 0**. This indicates that no core cells exist in the eight adjacent cells of the *ex-core* cell. This situation corresponds to the **dissipation** of a cluster.

2) ***Imp* = 1**. This signifies that the disappearance of the *ex-core* cell does not disrupt the density-reachability relationships among other core cells. However, the original cluster loses this cell, corresponding to the **reduction** of the cluster.

3) ***Imp* > 1**. This suggests that the disappearance of the ex-core cell may disrupt the density-reachability relationships among other core cells. As a result, the original cluster may split or merely shrink in size. This type of evolution is referred to as a **potential split**.

The DELETE process is similar to the INSERT process, with the primary difference being the case where $imp > 1$. In this situation, DELETE updates cluster labels (*cid*) iteratively using cell-BFS, starting from the local cell density-reachability of the adjacent cells. The number of cell-BFS operations is consistent with *imp*.



(a) Two clusters in LODC (b) Adjacent cells of Z (c) One cluster in LODC

Fig. 3. The update procedure in LODC. From (a)~(c), the p denotes a newly inserted point, whereas from (c)~(a), p denotes a newly deleted point.

Example 2: Figure 3(c)~3(a) illustrate a process of delete operation. Assume $MinPts$ of 1, which defines any cell with points as a core. In Figure 3(c), point p must be removed from *Cluster 3*. First, LODC executes the Collect-EX, updating the irradiated cells and collecting the *ExCoreCell* (*ExCoreCell* =

$\{Z\}$). Following this, the DELETE is executed to process cell Z. As shown in Figure 3(b), LODC identifies two local cell density-reachability relationships in the adjacent cells of Z, namely $\{X, X_4\}$ and $\{Y, Y_1, Y_4\}$, with an *imp* value of 2. Note that these cells still belong to the same cluster before deletion. Subsequently, LODC resets the *cid* of cell Z and performs cell-BFS from cells within each local density region. The process involves updating *cid* values iteratively for each cell-BFS operation. The detailed process is as follows: cell-BFS starts from X, visiting X_4, X_3, X_2, X_1, and updates the *cids* of these cells to 1. Next, cell-BFS starts from Y, visiting Y_1, Y_4, Y_2, Y_3, and updates the *cids* of these cells to 2. As a result, *Cluster 3* splits into two distinct clusters, *Cluster 1* and *Cluster 2* shown in Figure 3(a).

Time Complexity Analysis. The time complexity analysis for deletions is the same as that for insertions, $O(\bar{m})$. In practice, deletions often require more computational resources. This is primarily because deletions cannot directly determine splits. LODC uses a unified approach for split updates, which sometimes treats non-split cases (reductions) as splits, leading to longer deletion times.

5 Evaluation

To validate the lightweight online incremental density-based clustering algorithm, we conduct multiple experiments and primarily focus on the following two key aspects:

1) To evaluate the overall performance of LODC, we conduct experimental assessments from two aspects: update latency and cluster quality.

2) To examine the sensitivity of the LODC parameters, we adjust multiple parameters for the experiment.

5.1 Evaluation Settings

Competing Methods. LODC is compared with two other density-based incremental clustering algorithms: DenForest [12] and IncDBSCAN [5]. All algorithms use a unified sliding window model for consistent comparison, even though LODC is not constrained by this model.

Environment. Experiments are conducted on a laptop equipped with an Intel Core i7-12700H 14-core processor, using Python 3.8.

Datasets. One synthetic dataset (S2D) along with three real-world datasets (GeoLife [22], INTERACTION [20], and POI [3]) are utilized in experiments. All datasets are employed using their two-dimensional coordinate representations.

5.2 Overall Performance Evaluation

We evaluate the update latency and clustering quality of LODC under the parameters specified in Table 1. Unless otherwise specified, the cell side length is set to $\frac{\sqrt{2}}{2}\epsilon$, with a stride of 5% of the window size. Updates exceeding 20 minutes are excluded due to their impracticality in real-time scenarios.

Table 1. Threshold values and window sizes for each dataset.

Dataset	Distance (ϵ)	Density ($MinPts$)	$\|Window\|$ (Size)
S2D	0.075	4	10K
GeoLife	0.002	14	20K
INTERACTION	0.5	6	40K
POI	0.12	5	100K

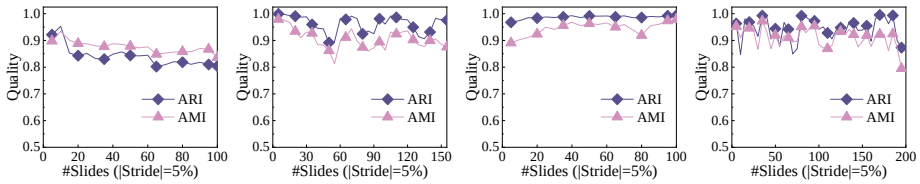
Update latency. As illustrated in Table 2, the total update latency of LODC is notably lower than that of the other two algorithms under evaluation. ***Insertion.*** LODC outperforms the other algorithms in handling insertions, primarily due to its avoidance of range searches, which leads to a significant improvement in speed. Additionally, LODC efficiently filters out some evolution types through *inf*. In scenarios where insertion updates are identified as emergence or expansion, LODC can complete the update in $O(1)$ amortized time. Cluster merging requires only a few cell-BFSs, resulting in minimal overhead. ***Deletion.*** The filtering mechanism employed by LODC also significantly reduces the overhead of deletion operations. Although DenForest demonstrates superior performance in deletion across three datasets, LODC achieves the best overall results in terms of total update latency in our experiments. LODC efficiently handles updates like merge and split through an innovative approach to cell processing. It significantly improves update latency over the DenForest and IncDBSCAN algorithms.

Table 2. Update latency over four datasets.

Dataset	Insert Latency (ms)			Delete Latency (ms)		
	LODC	DenForest	IncDBSCAN	**LODC**	DenForest	IncDBSCAN
S2D	**4**	87	9220	**4**	9	719000
GeoLife	**9**	1583	384031	**10**	7	Timeout
INTERACTION	**22**	1022	23021	**71**	32	Timeout
POI	**95**	10433	78906	**650**	56	Timeout

Clustering quality. We evaluate the clustering quality of LODC across four distinct datasets. True clustering labels are only available for the synthetic dataset; the other datasets lack ground truth labels. Therefore, we use DBSCAN's clustering results as a reference standard, applying the same parameters used with LODC. Clustering quality is assessed using the Adjusted Rand Index (ARI) [9] and Adjusted Mutual Information (AMI) [17] as evaluation metrics.

Figure 4 shows the clustering quality performance of LODC as the sliding window advances over time. The results demonstrate that LODC maintains a consistently high clustering quality across all four datasets as data is continuously updated. The average clustering quality measures are 0.87 for S2D, 0.93 for GeoLife, 0.951 for INTERACTION, and 0.900 for POI. LODC achieves relatively stable clustering results on S2D and INTERACTION. However, in GeoLife



(a) S2D (b) GeoLife (c) INTERACTION (d) POI

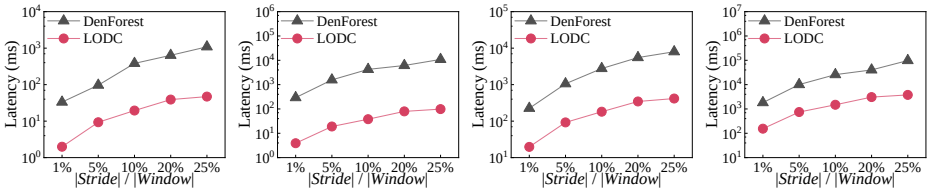
Fig. 4. Clustering quality on four datasets.

and POI, the clustering quality measures exhibit significant fluctuations. These fluctuations are primarily attributed to the skewness of data within the window.

5.3 Parameters Sensitivity Evaluation

This section compares the impact of various parameters on the performance of LODC. The experiments focus on three parameters: the window and stride size, the two density-based parameters ϵ & *MinPts* and the cell side length.

Varying size of window/stride. In the sliding window model, the window and stride sizes directly determine the scope of the data being processed and the interval for updating the analysis results. The distance and density thresholds are given in Table 1. Due to the update latency in most operations for IncDBSCAN exceeding 20 minutes, this algorithm is excluded from the experimental analysis.



(a) S2D (b) GeoLife (c) INTERACTION (d) POI

Fig. 5. Update latency with varying size of stride.

As illustrated in Figure 5 and Figure 6, LODC consistently exhibits significantly lower update latency than other clustering methods across all window and stride sizes settings. LODC demonstrates great robustness as the window or stride size increases. In the GeoLife dataset, each time the window size doubles, the update latency for LODC increases by factors of 2.13, 2.03, and 2.22, respectively. In contrast, the update latency for DenForest increases by factors of 2.58, 2.5, and 2.83. This suggests that LODC is capable of efficiently handling large-scale data scenarios that require real-time updates, with minimal performance degradation as the data volume increases. Overall, LODC outperforms

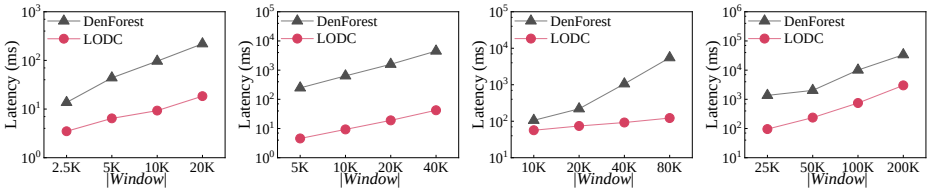


Fig. 6. Update latency with varying size of window ($|Stride|/|Window| = 5\%$).

other methods in minimizing update latency across varying window and stride sizes, demonstrating its robustness and suitability for real-time data processing.

Effect of density and distance thresholds. The distance threshold ϵ and the density threshold MinPts are the two core parameters in density-based clustering. We analyze the impact of these parameters on the S2D dataset. The window size used in the experiments is the same as shown in Table 1, with a stride size of 5% of the window size.

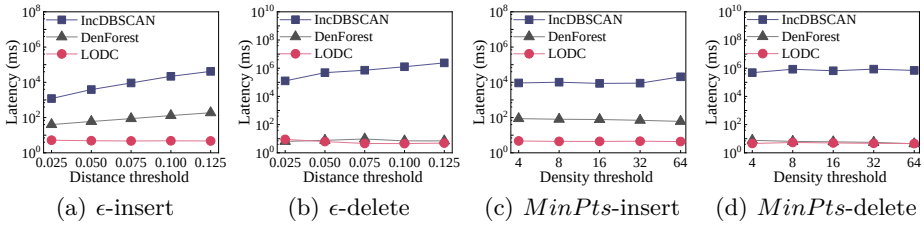


Fig. 7. Update latency with varying ϵ and $MinPts$ for the S2D dataset.

Figure 7(a) and Figure 7(b) illustrate the impact of different distance thresholds on the update latency for insertion and deletion operations, with the corresponding density thresholds consistent with those in Table 1. In traditional density clustering algorithms, the distance threshold directly determines the scope of range searches. However, since LODC does not require range searches, its update latency is almost unaffected by the distance threshold in both insertion and deletion operations, maintaining stability. In contrast, IncDBSCAN requires range searches in each update, causing a significant increase in update latency as the distance threshold increases. DenForest's insertion latency also shows an upward trend; however, its deletion operations rely on *DenTree* and do not involve range searches. As a result, its deletion latency remains largely unchanged. Figure 7(c) and Figure 7(d) show the effects of different density thresholds on update latency, corresponding to the distance thresholds provided in Table 1. As the figures demonstrate, the density threshold does not signif-

icantly affect the update latency for all three algorithms. LODC achieves the best experimental performance regarding insertion latency under various distance and density thresholds. Notably, LODC's deletion latency outperforms DenForest under some parameter settings, suggesting that further adjustments to these two fundamental parameters can further enhance overall efficiency.

Table 3. Latency and quality under four side lengths.

Dataset	Update Latency (ms)				Quality (ARI)			
	$1/4\epsilon$	$1/2\epsilon$	$\sqrt{2}/2\epsilon$	ϵ	$1/4\epsilon$	$1/2\epsilon$	$\sqrt{2}/2\epsilon$	ϵ
S2D	71	23	8	9	0.82	0.90	0.84	0.73
GeoLife	165	45	19	18	0.89	0.94	0.96	0.92
INTERACTION	5837	428	93	40	0.89	0.97	0.98	0.97
POI	1292	1190	746	444	0.31	0.85	0.91	0.70

Sensitivity analysis of cell side length. Cell size significantly impacts computational accuracy and efficiency. If cells are too small, LODC may revert to single-point computations. Conversely, overly large cells can obscure critical clustering information. To evaluate this, we analyze the impact of different cell sizes on time efficiency and clustering quality by assessing the performance of four datasets under various cell lengths. The experimental cell side lengths include $\frac{1}{4}\epsilon$, $\frac{1}{2}\epsilon$, $\frac{\sqrt{2}}{2}\epsilon$, and ϵ, with additional parameters detailed in Table 1.

Table 3 illustrates the change in update latency and quality across four datasets under different cell sizes. The results show that update latency decreases as cell sizes increase, aligning with our expectations. Clustering quality is generally higher for cell sizes of $\frac{1}{2}\epsilon$ and $\frac{\sqrt{2}}{2}\epsilon$. Surprisingly, the smallest cell size, $\frac{1}{4}\epsilon$, does not enhance clustering quality as expected. This phenomenon is due to the cell density-reachable strategy based on adjacent cells tending to form multiple small clusters under smaller cell sizes, which actually belong to a larger cluster, thus decreasing clustering quality. In summary, using $\frac{1}{2}\epsilon$ and $\frac{\sqrt{2}}{2}\epsilon$ as the initial grid sizes is a suitable choice.

6 Conclusion

This paper proposes a novel density-based incremental clustering algorithm, which significantly optimizes the efficiency of the clustering process and supports an insert-and-delete-friendly data processing mode. We delineate the specific procedural steps of our methodology and concurrently conduct an analysis of the computational complexity. Extensive experiments demonstrate that this algorithm achieves optimal performance in update latency and maintains high clustering quality as the data evolves. In future work, we intend to explore the application of the methodologies designed in this study within a parallel processing architecture, with the aim of further enhancing the system's throughput.

References

1. Besse, P.C., Guillouet, B., et al.: Review and perspective for distance-based clustering of vehicle trajectories. TITS **17**(11), 3306–3317 (2016)
2. Boonchoo, T., Ao, X., et al.: Grid-based dbscan: Indexing and inference. PR **90**, 271–284 (2019)
3. Center, S.: Map poi (point of interest) data. Peking University Open Research Data Platform (2017), https://doi.org/10.18170/DVN/WSXCNM
4. Ester, M., Kriegel, H.P., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. pp. 226–231 (1996)
5. Ester, M., Kriegel, H.P., et al.: Incremental clustering for mining in a data ware housing. In: VLDB. pp. 323–333 (1998)
6. Gan, J., Tao, Y.: Dynamic density based clustering. In: SIGMOD. pp. 1493–1507 (2017)
7. Hahsler, M., Bolaños, M.: Clustering data streams based on shared density between micro-clusters. TKDE **28**(6), 1449–1461 (2016)
8. Huang, X., Ma, T., et al.: Grit-dbscan: A spatial clustering algorithm for very large databases. PR **142**, 109658 (2023)
9. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification **2**, 193–218 (1985)
10. Jang, J., Jiang, H.: Dbscan++: Towards fast and scalable density clustering. In: ICML. pp. 3019–3029 (2019)
11. Kim, B., Koo, K., Kim, J., Moon, B.: Disc: Density-based incremental clustering by striding over streaming data. In: ICDE. pp. 828–839 (2021)
12. Kim, B., Koo, K., et al.: Denforest: Enabling fast deletion in incremental density-based clustering over sliding windows. In: SIGMOD. pp. 296–309 (2022)
13. Li, T., Huang, R., et al.: Compression of uncertain trajectories in road networks. PVLDB **13**(7), 1050–1063 (2020)
14. Reddy, K.S.S., Bindu, C.S.: Streamsw: A density-based approach for clustering data streams over sliding windows. Measurement **144**, 14–19 (2019)
15. Song, H., Lee, J.G.: Rp-dbscan: A superfast parallel dbscan algorithm based on random partitioning. In: SIGMOD. pp. 1173–1187 (2018)
16. Tu, L., Chen, Y.: Stream data clustering based on grid density and attraction. TKDD **3**(3), 1–27 (2009)
17. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: ICML. pp. 1073–1080 (2009)
18. Wang, Q., Chen, M., et al.: Detecting coherent groups in crowd scenes by multiview clustering. TPAMI **42**(1), 46–58 (2018)
19. Yang, D., Rundensteiner, E.A., Ward, M.O.: Neighbor-based pattern detection for windows over streaming data. In: EDBT. pp. 529–540 (2009)
20. Zhan, W., Sun, L., et al.: Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps. arXiv preprint arXiv:1910.03088 (2019)
21. Zheng, Y.: Trajectory data mining: an overview. TIST **6**(3), 1–41 (2015)
22. Zheng, Y., Fu, H., et al.: Geolife GPS trajectory dataset - User Guide, geolife gps trajectories 1.1 edn. (July 2011), https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/