DenForest: Enabling Fast Deletion in Incremental Density-Based Clustering over Sliding Windows

Bogyeong Kim
Seoul National University
Seoul, Korea
bgkim@dbs.snu.ac.kr

Kyoseung Koo
Seoul National University
Seoul, Korea
koo@dbs.snu.ac.kr

Undraa Enkhbat
Seoul National University
Seoul, Korea
undraae@dbs.snu.ac.kr

Bongki Moon
Seoul National University
Seoul, Korea
bkmoon@snu.ac.kr

ABSTRACT

The density-based clustering is utilized for various applications such as hot spot detection or segmentation. To serve those applications in real time, it is desired to update clusters incrementally by capturing only the recent data. The previous incremental density-based clustering algorithms often represent clusters as a graph and suffer serious performance degradation. This is because a costly graph traversal is required to check whether a cluster is still connected whenever a point is removed. In order to address the problem of slow deletion, this paper proposes a novel incremental density-based clustering algorithm called *DenForest*. By maintaining clusters as a group of spanning trees instead of a graph, *DenForest* can determine efficiently and accurately whether a cluster is to be split by a point removed from the window in logarithmic time. With extensive evaluations, it is demonstrated that *DenForest* outperforms the state-of-the-art density-based clustering algorithms significantly and achieves the clustering quality comparable with that of DBSCAN.

CCS CONCEPTS

• **Information systems → Clustering**; **Data stream mining**.

KEYWORDS

DenForest; Density-Based Clustering; Data Stream; Sliding Window

ACM Reference Format:
Bogyeong Kim, Kyoseung Koo, Undraa Enkhbat, and Bongki Moon. 2022. DenForest: Enabling Fast Deletion in Incremental Density-Based Clustering over Sliding Windows. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517833

1 INTRODUCTION

The density-based clustering method, which was pioneered by Ester *et al.* [15], is one of the most popular clustering approaches due to its unique characteristics. Applications that rely on density-based clustering include the detection of hot spots or segmented regions [16, 41, 52], geo-social network analysis [1, 32], the classification of LiDAR point clouds [10, 17], and mining events by clustering text messages [34]. The density-based clustering is, however, computationally intensive, and the execution of these analytic tasks for time-varying or streaming data involves significant challenges for real-time clustering.

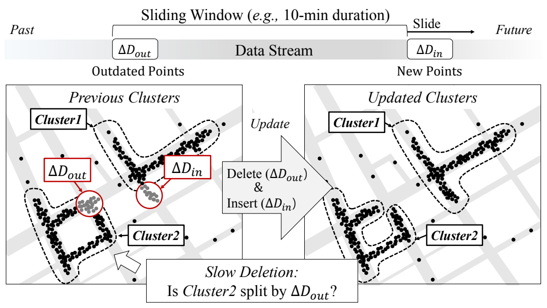


Figure 1: Density-Based Clustering over Sliding Windows

Consider, for example, a traffic monitoring system that periodically alerts the local public about congested regions. The congested regions (or density-based clusters) are determined based on the most recent ten-minute vehicular GPS data, which is updated every 30 seconds. The clusters will be reproduced periodically over the sliding window of a 10-min duration that advances every 30s. (See Figure 1 for illustration.) To perform this task in a timely manner, the density-based clustering method would update the congested regions incrementally rather than recomputing them from scratch every 30s. Such incremental clustering must update the congested regions efficiently by including new data points (ΔD_{in}) as well as excluding the outdated ones (ΔD_{out}) from the analysis.

The previous incremental density-based algorithms often manage clusters as a graph either *physically* or *logically*. Although representing clusters as a graph is a popular approach, it suffers serious performance degradation when a point is removed from its cluster. This is because deleting a point requires a costly graph traversal to check whether the cluster is still connected after the deletion. For example, the Incremental DBSCAN algorithm [14] requires numerous spatial range searches to traverse a graph whenever a point is removed. This task is essentially equivalent to the problem of dynamic graph connectivity [42], and it becomes the primary cause of the *slow deletion* by the incremental density-based clustering algorithms [48].

A recent approach called DISC [31] alleviates the performance degradation by minimizing the computational burden with batch operations. Nonetheless, DISC has the same time complexity as Incremental DBSCAN, and therefore it does not fundamentally address the problem. Another approach called Extra-N [51] avoids

This work was supported by the National Research Foundation of Korea (grant no. 2020R1A2C1010358). The authors assume all responsibility for the content of the paper.

Table 1: Time and Space Complexity

Method	Deletion	Insertion	Space
IncDBSCAN [14]	$O(N(N^{1-1/d}+k))$	$O(kN^{1-1/d}+k^2)$	$O(N)$
ExtraN [51]	-	$O(N^{1-1/d}+k(\frac{\lvert W\rvert}{\lvert S\rvert})^2)$	$O(N\frac{\lvert W\rvert}{\lvert S\rvert})$
ρ_2*-Approx* [21]	$O(log^2 N)$	$O(log^2 N)$	$O(NlogN)$
DISC [31]	$O(N(N^{1-1/d}+k))$	$O(kN^{1-1/d}+k^2)$	$O(N)$
DenForest	$\boldsymbol{O(logN)}$	$\boldsymbol{O(N^{1-1/d}+klogN)}$	$\boldsymbol{O(N)}$

N is the number of points within a sliding window, k is the number of points retrieved by a range search [2], and d is the dimensionality of data. |W| and |S| denote the size of a sliding window and its stride, respectively.

the slow deletion problem by pre-computing clusters in future windows, but it consumes memory too much and suffers from its own slow insert operation when clusters need to be updated frequently. Grid-based approximation methods have also been reported in the literature [20, 21]. The ρ-double-approximate DBSCAN algorithm [21] achieves poly-logarithmic time complexity for deletion by adopting Holm *et al.* 's data structure [26], which is proposed for dynamic graph connectivity algorithms. However, this grid-based algorithm requires a large number of approximate counting and nearest neighbor queries to such an extent that its practical performance becomes worse than the aforementioned approaches. The performance degradation is even more aggravated when high-resolution clusters are required [31, 40, 50].

The time and space complexity of the previous methods described above are summarized in Table 1. The Incremental DBSCAN and ρ-double-approximate DBSCAN algorithms are referred to as IncDBSCAN and ρ_2-Approx, respectively. Table 1 also includes the time and space complexity of *DenForest* we propose to address the problem of slow deletion squarely in this paper.

As an incremental density-based clustering algorithm, *DenForest* is based on a novel idea that allows us to manage clusters as a group of *spanning trees* of data points rather than a graph. In general, it is far simpler to determine whether the removal of a point splits a tree than a graph. However, a spanning tree being split does not always imply the underlying graph is also split. Therefore, we design a new data structure called *DenTree*, which can tell accurately whether the underlying graph is being split or not. The *DenTrees* of a graph can determine all by themselves whether the removal of a point splits the graph (*i.e.*, clusters). By managing clusters as *DenTrees*, *DenForest* addresses the slow deletion problem and achieves fast incremental density-based clustering.

The contributions of this work are summarized as follows.

- *DenForest* takes $O(logN)$ amortized time to delete a point, which is far faster than the other methods in comparison. Its performance is less sensitive to the dimensionality of data since it does not require range searches.
- *DenForest* takes $O(N^{1-1/d}+klogN)$ amortized time to insert a point. Although it is asymptotically slower than ρ_2-Approx, *DenForest* yields much higher performance in most practical settings.
- It is demonstrated through extensive experiments conducted on various real-world datasets that *DenForest* outperforms the currently available clustering methods considerably.
- It is confirmed by measuring the clustering quality in widely used metrics that the clustering quality of *DenForest* is not compromised, and *DenForest* and the DBSCAN algorithm are in fact comparable with respect to the clustering quality.

This paper is organized as follows. Section 2 presents the background of the traditional density-based clustering methods and introduces the task of clustering over sliding windows. Sections 3-4 present the proposed method, *DenForest*, and its incremental operations based on the two novel ideas, *nostalgic core* and *DenTree*. Section 5 analyzes the validity of the clusters from *DenForest*. The performance of *DenForest* is evaluated with various real-world datasets in Section 6. Lastly, some of the previous related studies are presented in Section 7.

2 BACKGROUND

2.1 Density-Based Clustering ($\mathcal{DC}$)

The density-based clustering algorithm DBSCAN was invented by Ester *et al.* a quarter century ago [15]. It detects clusters of arbitrary shapes even in the presence of noise. For a given set of data points, DBSCAN classifies them into three types, namely *core*, *border* and *noise* based on density (τ) and distance (ϵ) thresholds. A point p is classified as *core* if $|N_\epsilon(p)| \geq \tau$, where $N_\epsilon(p)$, known as the ϵ-neighbors of p, is the set of the neighboring points located within the ϵ-distance from p. A point q is classified as *border* if $|N_\epsilon(q)| < \tau$ but there is at least one *core* point in $N_\epsilon(q)$. All the other points are classified as *noise*.

DBSCAN defines a cluster as the maximal set of *core* and *border* points that are *density-reachable* from any *core* point in the cluster. A point p is said to be *directly density-reachable* from another point q if q is *core* and $p \in N_\epsilon(q)$. A point p is said to be *density-reachable* from another point q if there is a chain of *directly density-reachable core* points from q to p.

Reformulation of $\mathcal{DC}$. Given the way clusters are defined by DBSCAN, the density-based clustering ($\mathcal{DC}$) can be reformulated as the problem of finding the connected components in a graph [29, 35]. In the graph representation, each vertex corresponds to a data point and an edge is added to a pair of data points if they are within the ϵ-distance from each other. Each vertex is then labeled with one of *core*, *border*, or *noise* as described above. Finally, a connected component of *cores* as well as the *borders* adjacent to the connected component is identified as a cluster.

2.2 $\mathcal{DC}$ over Sliding Windows

The goal of this work is to find density-based clusters in streaming data under the sliding window model. The sliding window model is widely used to capture the recent state of streaming data, which cannot be stored in its entirety by virtue of the large and ever-increasing volume [11, 19].

2.2.1 ***Sliding Window Model.*** The sliding window model is characterized by two parameters known as *window* and *stride*. They are defined below for clarity.

Definition 1 (Window). *The window W is a set of the latest data points. The size of the window, $|W|$, can be bounded by either the number of points in the window (count-based window) or the duration of the window (time-based window).*

Definition 2 (Stride). *The stride S is defined by an interval at which the clustering result is updated while the window slides. The size of the stride, $|S|$, is bounded by the number of points or by a time duration depending on the type of the sliding window. The data points in the same stride are processed together.*

Suppose for example that the sliding window is time-based and its stride is set to 30 seconds. (The window size can be any time duration longer than 30 seconds.) Then, whenever the window slides by 30s, a group of new points may be added to the window during the period and they are processed together for updating clusters. Similarly, a group of old points may be removed from the window during the same period and they are also processed together for updating clusters.

2.2.2 ***$\mathcal{DC}$ over Sliding Windows.*** Since we aim at finding clusters in streaming data, we need to reproduce or update density-based clusters for the most recent data points captured in the current window, whenever the window strides. As is stated in Section 2.1, this task is equivalent to the problem of finding the connected components of cores and their adjacent borders *continuously* as the window advances.

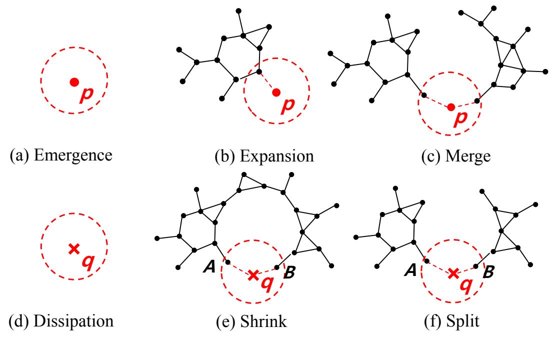


Figure 2: Cluster Evolution. Only the *cores* are shown in the figure. The point p denotes a newly added *core*, and the point q denotes a vanishing *core*.

There are six types of cluster evolution that can occur with a sliding window: *emergence, expansion, merge, dissipation, shrink* and *split*, as depicted in Figure 2. As new points are inserted into the window, some of the existing points may become *cores*. Due to those new cores, a new connected component (CC in short) may emerge or an existing CC may expand. If any of those new cores connects separate CCs, then they are merged into one. At the same time, old points may be deleted from the window, and some of the existing cores may become *non-cores*. Due to those vanishing cores, existing CCs may shrink or dissipate. A vanishing core may also split a CC into multiple CCs.

3 *DENFOREST*

This section presents a new incremental density-based clustering algorithm *DenForest* to deal with the slow deletion problem. First, an overview of the algorithm is given and the problem of slow deletion is stated formally. Then, a few key ideas such as *nostalgic cores* and *DenTree* are described in detail.

3.1 Overview of *DenForest*

Suppose data points are generated from sources such as sensing devices and are sent over for processing continuously. Each data point is associated with a *timestamp* T that indicates the event time or the ingestion time. The sliding window covers the most recent data points in the stream at any moment in time.

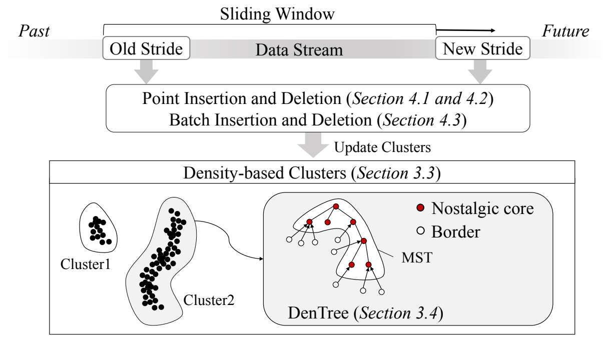


Figure 3: Overview of *DenForest*

For the data points in the current window, *DenForest* produces density-based clusters by detecting the connected components of *nostalgic cores* (Section 3.3). The *nostalgic cores* are similar to the cores defined by DBSCAN in that they are points found in the dense region. But the *nostalgic cores* differ from DBSCAN's cores in the way they expire and become non-core points. Each density-based cluster of *nostalgic cores* can be managed as a tree structure called *DenTree*, which can expedite the deletion process significantly (Section 3.4 and Theorem 1). When the window slides, *DenForest* updates clusters by inserting and deleting points individually (Sections 4.1 and 4.2) or in batch (Section 4.3). We assume that data points in the same stride are processed together, and data points in different strides are processed strictly in the order of their timestamps. The overall clustering procedure by *DenForest* is illustrated in Figure 3. The denotational symbols frequently used in the paper are summarized in Table 2.

Table 2: Notation

Symbol	Description
N	the number of points in the current window
τ, ϵ	the density and distance thresholds
CC	a connected component of *cores*
d-$core$	the *core* point defined by DBSCAN
T	the timestamp of a point
T_c	the core-expiration time of a point
$N_\epsilon(p)$	the neighboring points within ϵ-distance from p
$N'_\epsilon(p)$	the *previously inserted* points in $N_\epsilon(p)$
$\mathcal{MST}$	the maximum spanning tree of *nostalgic cores*
d	the number of dimensions
k	the number of points retrieved by a range query
M	the number of nodes in the Link-Cut tree
SC	*a super nostalgic core*
B_ϵ	a d-dimensional ball (or hypersphere) with radius ϵ
D_ϵ	the number of points in B_ϵ
$NC_\epsilon(p)$	the *nostalgic cores* within ϵ-distance from a *d-core* point p

Supported Types of the Sliding Window Model. *DenForest* supports both the count-based and the time-based sliding window models. First, under the count-based sliding window model, the stride size is one or greater. If the stride size is one, then a point insertion and a point deletion are performed alternately. If the stride size is greater than one, then a batch insertion and a batch deletion are performed alternately. The arrival order of data points in the stream can be used as the timestamp of an individual point. Second, under the time-based sliding window model, the stride size is a fixed time duration, and a batch insertion and a batch deletion are performed alternately and periodically. The only limitation is that the window size must be a multiple of the stride size for the batch-optimized operations to be applicable.

3.2 Slow Deletion Problem

The incremental management of clusters involves processing expiring (or vanishing) cores. This becomes a major bottleneck in updating density-based clusters incrementally.

Problem 1 (Slow deletion). *A cluster may be split by a vanishing core. To determine whether the cluster is split or not, the remaining cores need to be traversed to check the density-reachability from one another. This traversal would require a number of range searches, which is the main cause of the degraded performance of incremental density-based clustering.*

Example 1. *Consider a core point q that expires to become a non-core (either border or noise) point due to some other data points leaving the window. The vanishing core q will trigger one of the three types of evolution, namely dissipation, shrink, and split, as is shown in Figures 2(d)-(f). Let CC denote the connected component of q's cluster. Then a graph traversal such as Breadth-First Search (BFS) can be applied to check the connectedness of $CC \setminus \{q\}$. In Figures 2(e) and 2(f), the density-reachability between the two adjacent cores A and B need be checked when q vanishes. This will require visiting all the points in the figure inevitably. In general, if a graph traversal is initiated from one of the cores in $N_\epsilon(q)$ and it can visit all the cores in $N_\epsilon(q)$, then the cluster shrinks but does not split.*

This is equivalent to the dynamic connectivity problem, which has been studied for decades [26, 42]. Due to the $O(N^2)$ memory cost, the edges between core points are often maintained only logically. Thus, density-based clustering algorithms generally rely on a spatial index to discover a pair of adjacent cores. Hence, the latency concern is further aggravated due to the increasing cost of range searches, when the dimensionality of data points increases.

3.3 Nostalgic Core and Density-based Clusters

We come to realize that the slow deletion problem is caused intrinsically by the unpredictability of a vanishing core's expiration time. In this section, we present a novel approach that can precisely predict the expiration time of a core when it enters the window.

Similarly to the DBSCAN algorithm, *DenForest* adopts two parameters, namely density and distance thresholds (τ and ϵ) to discover density-reachable *cores* and adjacent *borders*. Unlike DBSCAN, however, *DenForest* relies on its own notion of a point being a core called a *nostalgic core* rather than that of DBSCAN. (DBSCAN's cores are referred to as *d-cores* hereinafter to distinguish one from another.) *DenForest* can determine exactly when a *nostalgic core p* will expire to become a *non-core* point, immediately after p enters the sliding window. This is done by considering only the current data points that entered the window earlier than p.

Definition 3 (Nostalgic core). *A point p in the window W is a nostalgic core if the number of ϵ-neighbors of p that entered W no later than p meets the density requirement. That is, p is a nostalgic core if $|N'_\epsilon(p)| \geq \tau$ where $N'_\epsilon(p) = \{q \in W \mid q \in N_\epsilon(p) \wedge q.T \leq p.T\}$, and $p.T$ and $q.T$ denote the timestamps of p and q, respectively.*

Whether a point p is a *nostalgic core* or not is determined at the insertion time solely by the existing points in the current window, and the core status of p is not affected by the points inserted in the future. Furthermore, when a *nostalgic core p* becomes a non-core point is also determined at the insertion time. (Refer to Lemma 1 below.) Note that DBSCAN's cores or *d-cores* do not possess any of these properties. While a point p stays in the window, DBSCAN allows p to gain or lose the core status at any time by pre-existing and future points leaving or entering the window. Note also that the set of *nostalgic cores* is always a *subset* of the set of *d-cores*.

Let $p.T_c$ denote the *core-expiration time* of p or the time when a *nostalgic core p* loses its core status to become a *non-core* point.

Lemma 1. *$p.T_c$ can be determined when p enters the window.*

Proof. Consider a point p that is about to enter the window. Assume $|N'_\epsilon(p)| \geq \tau$ and p is determined as a *nostalgic core.* Let q denote a point in $N'_\epsilon(p)$ such that its timestamp $q.T$ is the τ^{th} largest (or youngest). Then, p loses its core status when q leaves the window. Since q will leave the window at time $q.T + |W|$, p will become a non-core point at that time. That is, $p.T_c = q.T + |W|$. Therefore, the core-expiration time of p can be determined right at the moment when it enters the window. □

Lemma 2. *Once a point is not determined as a nostalgic core, then it can never become a nostalgic core until it leaves the window.*

Proof. For any point p in the window, $|N'_\epsilon(p)|$ can only decrease as the window slides. Therefore, if p is not a *nostalgic core* at the insertion time, it cannot become a *nostalgic core* until it leaves the window. □

Since *DenForest* defines its own *nostalgic cores*, the definitions of its border and noise points as well as its density-based clusters need to be altered accordingly.

Definition 4 (Border and noise of *DenForest*). *A point is a border if it is not a nostalgic core but within the ϵ-distance from any nostalgic core. Otherwise, it is considered a noise point.*

Definition 5 (Density-based cluster of *DenForest*). *In the graph representation (described in Section 2.1), a density-based cluster is defined as a connected component of nostalgic cores as well as the borders adjacent to the connected component.*

Each density-based cluster is managed as a tree called *DenTree* introduced in Section 3.4. It may appear that *DenForest* will produce density-based clusters of poor quality because the *nostalgic cores* are defined without considering the data points being inserted in the future. We will demonstrate later in the paper that *DenForest* can produce clusters of comparable quality much more efficiently. (See Sections 5 and 6.5 for the detailed evaluation.)

Cluster Membership of Border. Traditionally, the cluster membership of a border point is not decided deterministically. If a border point is adjacent to two or more clusters, it can join any of the clusters. This is the way DBSCAN decides the cluster membership of a border point. In contrast, we adopt a deterministic heuristic. *DenForest* attaches a border point p to a nostalgic core with the largest T_c among those in $N_\epsilon(p)$, which in turn decides the cluster membership of p. This approach is not in conflict with the definition of clusters and is in fact beneficial for performance, because deletion of the *nostalgic core* that p is attached to always results in p becoming *noise*, hence avoiding further reclassification effort.

3.4 *DenTree*

In order to process a point deletion efficiently, we maintain each cluster as a tree structure called *DenTree*, which can be used as an accurate barometer of a cluster split. A *DenTree* consists of a maximum spanning tree ($\mathcal{MST}$ in short) of a *DenGraph* defined below and border points associated with it.

Definition 6 (DenGraph). *A DenGraph $G(V, E, \mathcal{W})$ is an undirected edge-weighted graph where each vertex in V corresponds to a nostalgic core in the window, each edge in E corresponds to a pair of vertices within the ϵ-distance from each other, and each weight in $\mathcal{W}$ is set to the smaller T_c of the two adjacent vertices, namely*

$$\forall\, \overline{pq} \in E,\ w_{\overline{pq}} = \min\{p.T_c, q.T_c\}. \tag{1}$$

A *DenGraph* may have one or more $\mathcal{MST}$s, each of which corresponds to a connected component of the *DenGraph*. For a pair of *nostalgic cores* in the same connected component, there may exist multiple paths between them. Thus, just a path being split does not always make them disconnected in the graph. However, if the path being split is the one on the $\mathcal{MST}$ of the connected component, then the two *nostalgic cores* are no longer connected in the graph. This property of the $\mathcal{MST}$s is the key to addressing the slow deletion problem. Hereinafter, we refer to a maximum spanning tree of a *DenGraph* simply as an $\mathcal{MST}$ for brevity.

Theorem 1. *Consider two nostalgic cores p and q in an $\mathcal{MST}$ of a DenGraph. If another nostalgic core x on the path of the $\mathcal{MST}$ between p and q becomes a non-core point and is removed from the graph, then p and q are no longer connected not only in the $\mathcal{MST}$ but also in the graph.*

Proof. (By contradiction). Consider the moment when x is about to become a non-core point and be removed from the graph. Suppose the path on the $\mathcal{MST}$ passing through x is not the only path between p and q in the graph. Then there must be another path between them, and these two paths form a cycle. Since x is the one that is about to expire, all the points in the cycle have a *core-expiration time* greater than x's. Consider now a point y directly adjacent to x on the path of the $\mathcal{MST}$. Then, the edge $\overline{xy}$ must have the smallest weight among all the edges in the cycle, because $w_{\overline{xy}} = \min\{x.T_c, y.T_c\} = x.T_c$. This implies that the edge $\overline{xy}$ must not have been chosen for the $\mathcal{MST}$, and therefore contracts the assumption that $\overline{xy}$ is in the $\mathcal{MST}$. □

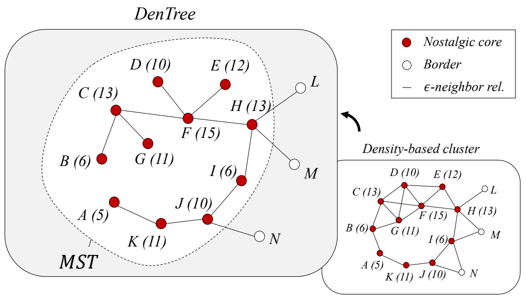


Figure 4: Example of *DenTree*

The implication of Theorem 1 is that when an $\mathcal{MST}$ is split by a vanishing nostalgic core, the underlying connected component (and its corresponding cluster) is also split. In general, a tree is split to non-empty subtrees when a node with two or more adjacent nodes is removed. Hence, to determine whether a cluster will be split by a vanishing nostalgic core p, it will be enough to check whether p is adjacent to two or more nostalgic cores in the $\mathcal{MST}$. Below we define *DenTree* that represents a cluster of nostalgic cores and border points.

Definition 7 (*DenTree*). *A DenTree is a tree composed of an $\mathcal{MST}$ and the border points associated with the $\mathcal{MST}$. If a border is adjacent to more than one nostalgic core, it is attached to the one of those that has the largest core-expiration time T_c.*

Example 2. *Figure 4 illustrates how a density-based cluster is represented by a DenTree, which consists of an $\mathcal{MST}$ of nostalgic cores and the border points associated with it. In the figure, the red points ($A \sim K$) and the white points ($L \sim N$) denote nostalgic cores and border points, respectively. An edge in the graph indicates that its two adjacent vertices (nostalgic cores or borders) are within the ϵ-distance from each other. Nostalgic cores are annotated with their core-expiration times. Unlike the traditional approaches, graph traversals are not required to determine whether a cluster is split or not by a vanishing core. For example, at time $t = 5$, point A becomes a non-core, and the cluster shrinks but is still connected because the DenTree is not split. On the other hand, at time $t = 6$, points B and I become non-cores, and the DenTree is split by point I, and consequently, the cluster is also split by point I.*

4 OPERATIONS OF *DENFOREST*

This section presents the detailed procedures of *DenForest*'s `Insert` and `Delete` operations. These procedures ensure that clusters produced by *DenForest* are always valid with respect to Definition 5, while the window slides.

4.1 Insertion

The `Insert` operation is responsible for updating clusters when new data points are added to the window. In particular, it ensures that $\mathcal{MST}$s are updated incrementally and remain valid even when the underlying graph of *nostalgic cores* changes over time by the sliding window. The overall procedure is composed of four steps as follows. (Refer to Algorithm 1 for details.)

Algorithm 1: Insert a point p and update the clusters

```
Insert (Point : p)
1  Insert p into the SpatialIndex
2  if |N'_ε(p)| ≥ τ then
3      mst ← 0 // The number of MSTs connected to p
4      p.T_c ← Core-Expiration-Time(p)
5      foreach n ∈ N'_ε(p) do
           // The Connect function returns true if it
              combines two disjoint MSTs.
6          if n.T_c ≥ currentTime and Connect(p, n) then
7              mst++
           end
       end
8      Determine the type of cluster evolution by the mst value
   end
9  Process the noise/borders
```

STEP 1 (Point Classification) First, it determines whether a new point p is a *nostalgic core* by counting the number of its ϵ-neighbors in the current window. If the count is no less than the density threshold τ, then p is classified as a *nostalgic core*. Otherwise, it is classified as a *border* or *noise* point.

STEP 2 (Determination of T_c***)*** If p is classified as a *nostalgic core*, the *core-expiration time* $p.T_c$ is computed by its ϵ-neighbors (Line 4). This step involves sorting the ϵ-neighbors in the order of their timestamps.

STEP 3 (Adding Links to $\mathcal{MST}s$***)*** If p is a *nostalgic core*, the maximum spanning trees ($\mathcal{MST}s$) are updated by adding p and new edges adjacent to it. The point p is connected to each of the *nostalgic cores* within the ϵ-distance by an edge whose weight is set by Equation (1). If a cycle is formed by adding a new edge, an edge with the smallest weight is removed from the cycle by the `Connect(p,n)` function (Line 6), which is described in Algorithm 2. Three types of cluster evolution can result by updating the $\mathcal{MST}s$ (Line 7 and Line 8):

(a) A cluster *emerges* if there is no $\mathcal{MST}$ near p ($mst = 0$).
(b) A cluster *expands* if p is connected to one $\mathcal{MST}$ ($mst = 1$).
(c) More than two clusters are *merged* when p is connected to multiple $\mathcal{MST}s$ ($mst \geq 2$).

STEP 4 (Updating Borders) If p is a *nostalgic core*, then an existing *border* point (say x) within the ϵ-distance from p may be reconnected to p, if T_c of p is greater than that of x's adjacent *nostalgic core*. For p that is not a *nostalgic core*, if there exists a *nostalgic core* in $N_\epsilon(p)$, then p becomes a *border* point and p is connected to a *nostalgic core* in $N_\epsilon(p)$ with the largest T_c. Otherwise, p becomes a *noise*.

The following lemma proves the validity of the `Insert` operation.

Lemma 3. `Insert(p)` *of Algorithm 1 updates DenTrees correctly.*

Proof. The $\mathcal{MST}s$ of a *DenGraph* remain cycle-free and maximally spanning because an edge with the smallest weight is removed if a cycle is formed by p being inserted [8]. Every border point, either a new or existing one, remains attached to a *nostalgic core* with the largest T_c within the ϵ-distance. Therefore, the *DenTrees* are updated correctly by `Insert`(p). □

Table 3: Link-Cut Tree Operations

APIs	Description
`Link(n,m)`	Link nodes n and m in different trees.
`Cut(n,m)`	Cut a link between nodes n and m.
`Connected(n,m)`	Check if a path exists between nodes n and m.
`FindMinE(n,m)`	Find the minimum weighted edge on the path between nodes n and m (added for *DenForest*).

4.1.1 $\mathcal{MST}$ based on Link-Cut Tree.

DenForest relies on a data structure called *Link-Cut Tree* [44] to efficiently detect and break a cycle in the $\mathcal{MST}s$. The *Link-Cut tree* represents a set of trees and is often used to solve the dynamic connectivity problem for an acyclic graph. The trees in a *Link-Cut tree* are divided into disjoint paths, and each path is represented by a Splay tree. By managing a set of trees with a path-based structure, the *Link-Cut tree* can support its key operations in the amortized $O(\log M)$ time, where M is the total number of nodes in the trees. See Table 3 for the list of supported operations as well as a new one added for *DenForest*.

DenForest maintains its $\mathcal{MST}s$ in the *Link-Cut tree* and updates them whenever the underlying graph changes by a new point added to the window. For efficient updates, we design a new function called `FindMinE(n,m)` in addition to the traditional operations of the Link-Cut tree. `FindMinE(n,m)` finds the minimum weighted edge on the path between two nodes n and m in a tree. `FindMinE(n,m)` also runs in the amortized $O(\log M)$ time.

Algorithm 2: Connect two points in the Link-Cut tree

```
Connect (Point : p, Point : q)
1  w_pq ← min{p.T_c, q.T_c}
2  if Connected(p,q) then
       // If a cycle is formed, cut the minimum weighted edge
3      rs ← FindMinE(p, q)
4      if w_rs ≤ w_pq then
5          Cut(r, s) and Link(p, q)
       end
6      return False // No merge
   else
7      Link(p, q)
8      return True // Potential merge
   end
```

Algorithm 2 presents the `Connect` algorithm that links two *nostalgic cores* p and q in the *Link-Cut tree.* The weight of the edge $\overline{pq}$ is set by the smaller of the *core-expiration times* of p and q (Line 1). If there already exists a path between them (Line 2), adding $\overline{pq}$ would create a cycle. Thus, the algorithm finds an edge with the smallest weight, say $\overline{rs}$, on the path between p and q (Line 3). If the weight of $\overline{rs}$ is smaller than the weight of $\overline{pq}$, then $\overline{rs}$ is removed and replaced by $\overline{pq}$ (Lines 4-5). Otherwise, $\overline{pq}$ is simply dropped without altering the Link-Cut tree (Line 6). If there is no path between p and q, the two separate $\mathcal{MST}s$ they belong to are linked together by adding $\overline{pq}$ (Line 7). The `Connect` algorithm returns a Boolean flag to indicate the type of cluster evolution.

4.1.2 ***Time Complexity of*** **`Insert`** ***Operation.*** The runtime of the `Insert` algorithm is given by the formula below.

$$C_i + C_r + P_{nc} \times (C_s + |N'_\epsilon| C_c) + C_p$$

C_i is the cost of inserting a point into the spatial index, C_r is the cost of a range search, P_{nc} is the probability of an inserted point being a *nostalgic core*, C_s is the cost of sorting N'_ϵ to compute the *core-expiration time*, C_c is the cost of the `Connect` operation, and C_p is the cost of processing a *border*.

Lemma 4. `Insert` *runs in amortized* $O(N^{1-1/d} + k \log N)$ *time.*

Proof. Assume that the balanced *k-d tree* [2] is used as a spatial index. Then, C_i and C_r are $O(\log N)$ and $O(N^{1-1/d}+k)$, respectively, where k is the number of points retrieved by a range query. C_s and C_p are $O(|N'_\epsilon| \times \log |N'_\epsilon|)$ and $O(|N'_\epsilon|)$, respectively. C_c is amortized $O(\log M)$ because all of its sub-algorithms take amortized $O(\log M)$ time. Then, since $M < N$, the amortized time complexity of the `Insert` operation is bounded by $O(N^{1-1/d} + k \log N)$. □

DenForest can work with many spatial indexes such as *R-tree* [24] and *range tree* [3] as well. We only assume the balanced *k-d tree* in the proof for its well known upperbound analysis.

4.2 Deletion

The `Delete` operation is responsible for updating clusters when existing data points are removed from the sliding window. Theorem 1 enables it to quickly determine whether a cluster will be split or not just by counting the links adjacent to each vanishing *nostalgic core* in the $\mathcal{MST}$. The `Delete` algorithm depicted in Algorithm 3 runs in two main steps.

Algorithm 3: Delete a point q and update the clusters

```
Delete (Point : q)
1  E(q) : a set of nostalgic cores expired by the deletion of q
2  foreach x ∈ E(q) do
3      L ← a set of nostalgic cores linked to x
4      Determine the type of cluster evolution by the |L| value
5      foreach y ∈ L do Cut(x,y)
6      Reclassify x as either border or noise by the |L| value.
   end
7  Delete q from the SpatialIndex
```

STEP 1 (Finding Expiring Nostalgic Cores) When a point q is removed from the window, some of the *nostalgic cores* may become *non-cores*. Unlike the traditional density-based methods, those expiring *nostalgic cores* can be found without executing any range search. Since the *core-expiration time* of a *nostalgic core* is determined at the insertion time and remains intact, all the *nostalgic cores* in the current window can be indexed in a supplementary data structure such as `hashmap` with their *core-expiration times* as keys. When q is removed from the window at time t, all the *nostalgic cores* becoming non-cores at time t can be found in $O(|E(q)|)$ time from the supplementary data structure (Line 1).

STEP 2 (Cutting Links from $\mathcal{MST}$***s)*** All the expiring *nostalgic cores* are examined to determine whether any $\mathcal{MST}$ is to be split. For each expiring *nostalgic core* x, all the adjacent links are found (Line 3) and removed (Line 5). Then x is reclassified as follows (Line 6). If $|L| \geq 1$, then x becomes a *border* point and is attached to a *nostalgic core* in L with the largest T_c. If $|L| = 0$, then there is no *nostalgic core* adjacent to it and x becomes a *noise*. Three types of cluster evolution can result from each expiring *nostalgic core* x (Lines 4):

(a) If $|L| = 0$, the cluster containing x dissipates.
(b) If $|L| = 1$, the cluster containing x shrinks.
(c) If $|L| \geq 2$, the cluster containing x is split.

Theorem 1 guarantees that the clusters updated by the `Delete` operation are valid. Note also that Algorithm 3 does not involve any range search. Consequently, the performance of the `Delete` operation is less sensitive to the dimensionality of data points, and is not overly affected by the distance threshold ϵ. This will be corroborated by the experimental evaluation in Section 6.4.

4.2.1 ***Time Complexity of*** **`Delete`** ***Operation.*** The asymptotic runtime of the `Delete` algorithm is given by the formula below.

$$O(|E| \times |L| \times \log M + \log N) \tag{2}$$

$|E|$ is the number of *nostalgic cores* expired by the deletion of a point, and $|L|$ is the degree (*i.e.*, the number of adjacent links) of an expiring *nostalgic core.* The terms $\log M$ and $\log N$ denote the cost of a `Cut` operation in the Link-Cut tree and the cost of a deletion in the spatial index, respectively.

In the following theorem that bounds the runtime of the `Delete` algorithm, we assume that an unbounded number of data points can take up exactly the same location in space. We then determine the maximum number of *nostalgic cores* that can expire by a single point being removed from the sliding window.

Theorem 2. *The amortized runtime of the* `Delete` *algorithm is* $O(\log N)$ *where* N *is the number of points in the sliding window.*

Proof. Suppose a point p is about to be removed from the window and a *nostalgic core* x is about to become a *non-core* by that. First, x must be in $N_\epsilon(p)$. Otherwise, x would not be affected by the deletion of p. Second, the number of points that exist at exactly the same location as x must be no more than $\tau - 1$. Otherwise, x would still be a *nostalgic core* after the deletion. Third, if there are $\tau - 1$ points at the location of x, then there must be no more point within the ϵ-distance from x. Otherwise, again, x would still be a *nostalgic core* after the deletion. That is, if $\tau - 1$ *nostalgic cores* at the same location are about to expire, there must be no other point within the ϵ-distance. Hence, the number of those locations where groups of $\tau - 1$ expiring *nostalgic cores* coexist is bounded by a constant. In a 2-dimensional space, the maximum number of such locations is six, which is known as a *kissing number* [9].[1] In a d-dimensional space, the kissing number is bounded by c^d, where c is a small constant. Thus, the number of *nostalgic cores* expired by the deletion of p is bounded by $(\tau - 1) \times c^d$, which is $O(1)$. This implies that $|E|$ is $O(1)$ in Equation (2). Besides, in Equation (2), the average of $|L|$ is less than two (just like the average degree of a vertex in any tree or acyclic graph), and $M \leq N$ because M is

[1]The kissing number is defined as the maximal number of non-overlapping unit spheres that can touch a common sphere of the same size. For *DenForest*, the radius of the spheres is $\epsilon/2$.

the number of nodes in the Link-Cut tree. Therefore, the amortized runtime of the `Delete` algorithm is $O(\log N)$. □

Lemma 5. *DenForest consumes O(N) space.*

Proof. The main data structures *DenForest* relies on are *DenTrees*, a spatial index and a hashmap. *DenTrees* including the *Link-Cut* tree use $O(N)$ space, and the spatial index and the hashmap both use $O(N)$ space. □

Cluster Membership. *DenForest* does not store the cluster identification of an individual point. If it did, then the cost of updating the cluster identifications would be non-trivial. Instead, upon request, *DenForest* assigns a unique ID to the points belonging to each cluster by traversing the corresponding *DenTree*. This procedure requires $O(N)$ time, which is no worse than any existing method.

4.3 Batch-Optimized Update

The `Insert` and `Delete` operations can be further optimized by exploiting the locality of the data points in the same stride. By consolidating nearby *nostalgic cores* to fewer meta-objects called *super nostalgic cores*, *DenForest* can make the $\mathcal{MST}s$ smaller and reduce the overhead of updating clusters.

Definition 8 (Super nostalgic core, SC). *The super nostalgic core is a connected component of nostalgic cores in the same stride that become non-cores together when the window slides by a single stride.*

Definition 9 (ϵ-Neighbors of a super nostalgic core). *Two super nostalgic cores sc_1 and sc_2 are said to be ϵ-neighbors if there are a pair of points $p \in sc_1$ and $q \in sc_2$ such that $distance(p, q) \leq \epsilon$.*

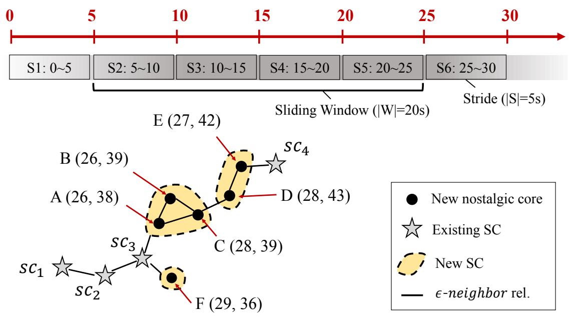


Figure 5: Example of Super Nostalgic Cores

In the following example of *super nostalgic cores*, we assume that the window and the stride are 20 seconds long and 5 seconds long, respectively, and the sliding window is currently anchored at time 25 covering a time interval (5,25]. This is illustrated in Figure 5, where each point is annotated with the timestamp (T) and the core-expiration time (T_c). For example, point $A(26, 38)$ will be ingested at time 26 and will become a *non-core* at time 38. *Non-cores* are not shown in the figure.

Example 3. *In Figure 5, the four stars ($sc_1 \sim sc_4$) represent super nostalgic cores in the current window. When the window advances by a stride, six new points ($A \sim F$) in the stride S6 are added to the window, and they all become nostalgic cores. Among those six points, F is separate from the others by a nostalgic core not in stride S6, and it alone forms a super nostalgic core $\{F\}$. The other points $A \sim E$ are in a connected component of nostalgic cores in the same stride, but they form two separate super nostalgic cores $\{A, B, C\}$ and $\{D, E\}$. This is because points A, B, and C will become non-cores at time 40, while points D and E will become non-cores at time 45. The super nostalgic cores $\{A, B, C\}$ and $\{D, E\}$ are said to be ϵ-neighbors because C and D are within the ϵ-distance from each other.*

The **batch-optimized** `Insert` algorithm replaces a group of connected *nostalgic cores* with a *super nostalgic core* so that *DenForest* can update clusters more efficiently without compromising the clustering result. The detailed procedure is given below.

STEP 1 (Finding SCs) When the window slides by a stride and new data points are added, new *nostalgic cores* are found from the data points, and new *super nostalgic cores* are formed from the *nostalgic cores*. The *core-expiration time* of a *super nostalgic core* is set to the time interval of a stride that covers all the *core-expiration times* of its *nostalgic cores*. For example, T_c of the *super nostalgic core* $\{A, B, C\}$ is set to the time interval (35,40]. Besides, each *nostalgic core* maintains a pointer to the adjacent *nostalgic core* with the largest T_c for the batch-optimized deletion. For example, point C maintains a pointer to D, which has the largest T_c among $N_\epsilon(C)$.

STEP 2 (Updating $\mathcal{MST}s$ with SCs) Each *super nostalgic core* is collapsed to a single vertex in the $\mathcal{MST}s$. A new edge is introduced to each pair of the ϵ-neighbors of *super nostalgic cores*, and the weight of the new edge is set to the smaller of their *core-expiration times*. For example, the weight of the edge between two *super nostalgic cores* $\{A, B, C\}$ and $\{D, E\}$ is set to the time interval (35, 40]. If a cycle is formed, then an edge with the smallest weight is removed from the cycle. *Borders* are updated the same way as the `Insert` algorithm.

The **batch-optimized** `Delete` algorithm works similarly. When the window slides by a stride and old data points are removed, some of the *super nostalgic cores* may become *non-cores*. For example, when the window slides from a time interval (15, 35] to a time interval (20, 40], *super nostalgic cores* $\{A, B, C\}$ and $\{F\}$ become *non-cores*. The expired *super nostalgic cores* are removed from the $\mathcal{MST}s$. For each point p that has become a *non-core*, the adjacent *nostalgic core* q with the largest T_c is examined, which is found by following the pointer established in the batch insertion algorithm. If q is still a *nostalgic core*, then p becomes a *border* point. Otherwise, it becomes a *noise*.

5 CLUSTERING QUALITY OF *DENFOREST*

Two aspects should be considered in evaluating the effectiveness of a clustering method for streaming data over the sliding window. The first is the capability to produce *high-quality* clusters from the current window, and the second is the capability to sustain the quality *efficiently* while the window moves forward. This section evaluates the first aspect of *DenForest*. The second aspect of *DenForest* will be evaluated in Section 6.

5.1 Clustering Quality for Static Data

A variety of synthetically generated labeled datasets were used for the evaluation of clustering quality. For each dataset, it was assumed that the entire set of data points were contained in the current window from which density-based clusters were produced

Table 4: Clustering quality on various datasets

Dataset	*DenForest* vs. Label			DBSCAN vs. Label			*DenForest* vs. DBSCAN		
	ARI	AMI	NMI	ARI	AMI	NMI	ARI	AMI	NMI
Spiral [6]	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
*R*15 [47]	0.98	0.98	0.98	0.99	0.99	0.99	0.98	0.98	0.98
Aggr. [22]	0.97	0.96	0.97	0.99	0.99	0.99	0.97	0.96	0.97
Comp. [53]	0.94	0.87	0.90	0.94	0.85	0.91	0.98	0.88	0.94
*G*2-2-30 [18]	0.95	0.88	0.90	0.96	0.93	0.93	0.96	0.92	0.94
*G*2-4-30 [18]	0.99	0.97	0.99	1.00	1.00	1.00	0.99	0.97	0.99
*G*2-8-30 [18]	0.99	0.99	0.99	1.00	1.00	1.00	0.99	0.99	0.99
Average	**0.97**	**0.95**	**0.96**	**0.98**	**0.96**	**0.97**	**0.98**	**0.95**	**0.97**

by *DenForest* as well as DBSCAN for comparison. *DenForest* produces clusters of *nostalgic cores*, while DBSCAN produces clusters of *d-cores* (*i.e.*, cores in DBSCAN's own definition). Although they define *cores* in their own ways, both *DenForest* and DBSCAN define *clusters* the same way.

We adopted three metrics called Adjusted Rand Index (ARI) [27], Adjusted Mutual Information (AMI) [49], and Normalized Mutual Information (NMI) [33] to measure the clustering quality quantitatively. These metrics have been used widely in various studies to compute the similarity between two cluster memberships (or partitions) [5, 30]. The ARI values range from -1 to 1, with 1 indicating two identical clustering results and -1 indicating no similarity between them. The AMI and NMI are similar to ARI, but their values range from 0 to 1. For each clustering method, we attempted to obtain the best achievable quality by tuning the density (τ) and the distance (ϵ) thresholds. The clustering results of *DenForest* may vary depending on the ingestion order of data points owing to the way *nostalgic cores* are defined. Thus, for each metric, we ran *DenForest* one hundred times for each dataset each with a random ingestion order and took the average.

Table 4 summaries the clustering quality of *DenForest* and DBSCAN tested on seven labeled datasets, which are listed in the first column of the table. In the first and second groups of three columns are the quality measurements computed with respect to the given labels (*i.e.*, ground truth), which measure the ability of *DenForest* and DBSCAN to produce accurate clustering results. In the third group of three columns are the quality measurements computed with respect to the clustering results from DBSCAN, which measures the ability of *DenForest* to produce the same clustering results as those of DBSCAN. On average, *DenForest* achieved 0.97 (ARI), 0.95 (AMI), and 0.96 (NMI) clustering quality with respect to the given true labels, and achieved 0.98 (ARI), 0.95 (AMI) and 0.97 (NMI) clustering quality with respect to the clustering results from DBSCAN. This demonstrates that considering only the pre-existing data points in the current window does not overly compromise the quality of clusters and helps expedite the clustering process significantly, which will be shown in Section 6.

5.2 Replaceability

The number of *nostalgic cores* within the distance threshold is critical to the quality of clustering result. This section provides further analysis on the relationship among the density of a region, the number of *nostalgic cores*, and the clustering quality. We will first show that the number of *nostalgic cores* in a region is linearly correlated with the density of the region (Section 5.2.1). We will then show that *DenForest* and DBSCAN would produce similar clustering results if the region was dense enough (Section 5.2.2).

5.2.1 *Nostalgic Cores and Density.*

Imagine a set of points scattered in the space and time. The number of points in a space V and a time-interval $(t_1, t_2]$ can be calculated by the equation below with continuous density assumed for simplicity.

$$\int_{t_1}^{t_2} \int_V D(\vec{x}, t) \mathrm{d}V \mathrm{d}t \tag{3}$$

where $D(\vec{x}, t)$ denotes the density function of space ($\vec{x}$) and time (t). Below we define a *locally stable* subspace whose number of *nostalgic cores* can be determined with respect to the density of the subspace.

Definition 10 (**Locally stable**). *A subspace is said to be locally stable if* $\int_{B_\epsilon(p)} D(\vec{x}, t)\mathrm{d}V = Vol(B_\epsilon) \cdot D(p, t)$ *for any point* p *in the subspace.* B_ϵ *and* $B_\epsilon(p)$ *denote a ball of radius* ϵ *and a ball of radius* ϵ *centered at* p*, respectively.* $Vol(B_\epsilon)$ *denotes the volume of* B_ϵ.

Lemma 6. *In a locally stable subspace, the number of nostalgic cores in any* B_ϵ *is* $D_\epsilon - \tau$*, where* D_ϵ *denotes the number of points in* B_ϵ.

Proof. Let the time interval of the window be $(t_0, t_W]$. D_ϵ is equal to $\int_{t_0}^{t_W} \int_{B_\epsilon} D(\vec{x}, t)\mathrm{d}V\mathrm{d}t$ by Equation (3). For a position $\vec{y}$ in the *locally stable* space, define a function $T(\vec{y})$ such that $\tau = \int_{t_0}^{T(\vec{y})} \int_{B_\epsilon(\vec{y})} D(\vec{x}, t)\mathrm{d}V\mathrm{d}t$. $T(\vec{y})$ is a time threshold for *nostalgic core* classification. Among those at the same location as $\vec{y}$, the points inserted after the $T(\vec{y})$ time are classified as *nostalgic cores*, while the points inserted before that time are not. The number of *nostalgic cores* in B_ϵ can then be defined as $\int_{B_\epsilon} \int_{T(\vec{x})}^{t_W} D(\vec{x}, t)\mathrm{d}t\mathrm{d}V$. Therefore, the lemma is proved as follows.

$$\begin{aligned}
&\int_{B_\epsilon} \int_{T(\vec{x})}^{t_W} D(\vec{x}, t)\mathrm{d}t\mathrm{d}V = D_\epsilon - \int_{B_\epsilon} \int_{t_0}^{T(\vec{x})} D(\vec{x}, t)\mathrm{d}t\mathrm{d}V \\
=&D_\epsilon - \int_{B_\epsilon} \tau / Vol(B_\epsilon)\mathrm{d}V \qquad (\textit{by the locally stable condition}) \\
=&D_\epsilon - \tau.
\end{aligned}$$

□

This lemma indicates that in the region with a sufficiently high density ($D_\epsilon \gg \tau$), there will be many *nostalgic cores* in any B_ϵ.

5.2.2 *Nostalgic Cores and Quality.*

The *d-cores* of DBSCAN play two important roles : (1) spatially *covering* the clustered region and (2) *connecting* the neighboring points. If *nostalgic cores* completely *replace d-cores* playing these roles, *DenForest* and DBSCAN will produce an identical result. This replaceability is correlated with the number of *nostalgic cores* in B_ϵ.

For a *d-core* point p, let $NC_\epsilon(p)$ be a set of *nostalgic cores* in $B_\epsilon(p)$. Recall that *nostalgic cores* are a subset of *d-cores*.

Definition 11 (**Completely replaceable**). *A d-core p is said to be completely replaceable by* $NC_\epsilon(p)$*, if the following conditions are satisfied.*

$$B_\epsilon(p) \subseteq \bigcup_{q \in NC_\epsilon(p)} B_\epsilon(q) \qquad (\textit{Coverage})$$

A DenGraph's subgraph whose vertex set is $NC_\epsilon(p)$ *is connected.* $\qquad$ (*Connectivity*)

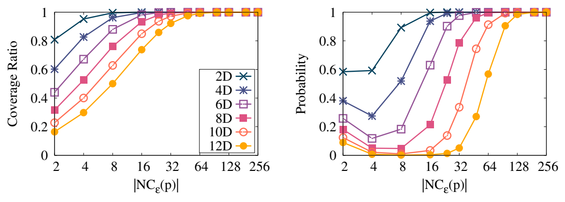
Theorem 3. *For any d-core p, if it can be completely replaced by $NC_\epsilon(p)$, then both DenForest and DBSCAN produce an identical clustering result.*

Proof. The *Coverage* condition guarantees that the area covered by *nostalgic cores* contains all the borders and *d-cores* of DBSCAN. The *Connectivity* condition guarantees that all the *d-cores* in a cluster of DBSCAN are included in a cluster of *DenForest*. □

This theorem clearly states that the two conditions of Definition 11 are relevant to clustering quality. Now, let us find out how they are correlated with the cardinality of $NC_\epsilon(p)$. Figure 6 shows the coverage ratio of a point p and the probability of the subgraph composed of $NC_\epsilon(p)$ being connected, with respect to the $|NC_\epsilon(p)|$ and the dimensionality of space. We adopted the *Monte Carlo* method [36], and $NC_\epsilon(p)$ is populated uniformly around p. The coverage ratio is calculated by the following equation.

$$Coverage\ Ratio = \frac{Vol(B_\epsilon(p) \cap (\bigcup_{q \in NC_\epsilon(p)} B_\epsilon(q)))}{Vol(B_\epsilon)}$$

The general trend is that, as $|NC_\epsilon(\mathrm{p})|$ increases, both the coverage and the connectivity increase. For example, in a 2D space, if $D_\epsilon - \tau \geq 16$, then the clustering result of *DenForest* will be nearly identical to that of DBSCAN. This is because 16 or more *nostalgic cores* around a *d-core p* can replace it completely.



(a) Coverage **(b) Connectivity**

Figure 6: Coverage and Connectivity w.r.t. $|NC_\epsilon|$

In *interior regions* of a cluster where the density is sufficiently high ($D_\epsilon \gg \tau$), there would be many *nostalgic cores* within the ϵ-distance. Thus, the *nostalgic cores* would replace *d-cores* well with the high coverage ratio and the high probability of being connected. In *boundary regions* of a cluster where the density is not so high ($D_\epsilon \approx \tau$), there might not be enough points in NC_ϵ, and the *nostalgic cores* would not replace *d-cores* so well. However, the boundary regions are fundamentally unstable, and they seldom affect the quality of clustering result.

6 EVALUATION

This section analyzes the performance of *DenForest* by comparing it with the existing incremental density-based clustering methods.

Competing Methods. *DenForest* is compared with three incremental methods that can produce exactly the same clustering result as that of DBSCAN : Incremental DBSCAN, Extra-N, and DISC. Incremental DBSCAN (or IncDBSCAN in short) is an incremental version of DBSCAN that supports the insertion and deletion of an individual data point [14]. We used its version optimized with MS-BFS [31]. Extra-N [51] is another clustering method that supports incremental updates under the sliding window model. DISC [31] is a recent one that can expedite the processing of incremental operations by performing them in batch.

DenForest is also compared with ρ-double-approximate DBSCAN that produces an approximate clustering result [21]. Two approximation parameters were chosen in the experiments, ρ=0.001 and ρ=0.1, for nearly accurate and less accurate clusters, respectively. The clustering results produced with these two parameters are denoted by *Approx-High* (ρ=0.001) and *Approx-Low* (ρ=0.1). Finally, the clustering results produced by *DenForest* without the batch-optimization (presented in Section 4.3) is denoted by *DenForest-NO*.

Environment. All the experiments were conducted on a stand-alone machine with a Ryzen 7 1700 8-Core Processor, 64 GB RAM, and a 256 GB solid-state drive, running Ubuntu 18.04 LTS. We implemented all the clustering methods in comparison as well as the R-tree spatial index in Java with JDK 1.8.0. The elapsed times were measured using the `System.nanoTime` function. Since each dataset was preloaded into the memory, the disk did not affect the performance during the experiments.

Real-World Datasets. In the experiments, the following four real-world datasets were used to evaluate the proposed method.

DTG is a dataset collected from digital tachograph devices attached to commercial vehicles in a metropolitan city [12]. A record was generated from each vehicle every 10 seconds, and each record included the time, location, speed, and acceleration of the vehicle. The 2D coordinates (p_{lat}, p_{lon}) were used in the experiments, where p_{lat} and p_{lon} are the latitude and the longitude fields, respectively. The total number of records is approximately 300 million.

GeoLife is a GPS trajectory dataset collected from 182 users over a period of four years [54]. Each record includes the time and the location of each user. The 3D normalized coordinates $(p_{lat}, p_{lon}, p_{alt}/300,000)$ were used in the experiments, where p_{alt} is the altitude field. The total number of records is approximately 24.8 million.

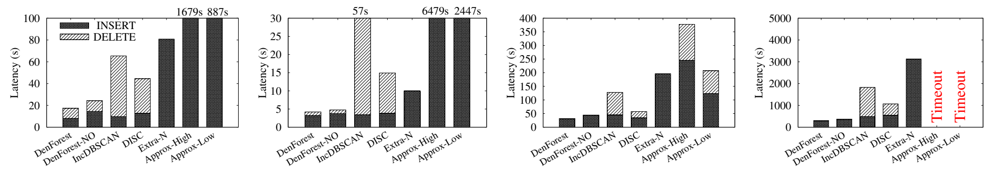
IRIS is a dataset of earthquake events that occurred around the world from 1960 to 2019 [28]. The 4D normalized coordinates $(p_{lat}, p_{lon}, p_{dep}/10, p_{mag}\times 10)$ were used in the experiments, where p_{dep} and p_{mag} are the depth and the magnitude fields, respectively. The total number of records is approximately 1.8 million.

Household is a dataset of the electric energy consumption in a household over a period of four years [13]. Each record includes seven fields related to the power and voltage information. The 7D coordinates normalized by the variance of the fields were used in the experiment. The total number of records is approximately 2 million.

6.1 Evaluation Settings

Sliding window model. The clustering methods were evaluated under the *count-based* sliding window model, where the *window* and the *stride* are sized by the number of data points. This is because the count-based model is easier to control the workloads. Nonetheless, the ingestion order of data points still follows their timestamps. The default window size was set to a fraction of each dataset, roughly corresponding to a chosen time duration.

Parameters. The density (τ) and the distance (ϵ) thresholds of all the methods were set according to the following scheme. For



(a) DTG **(b) GeoLife** **(c) IRIS** **(d) Household**

Figure 7: Update Latency (|Stride|/|Window|=5%)

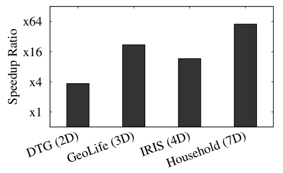


Figure 8: Speedup of `Delete`

the DTG and GeoLife datasets, a traffic monitoring example was adopted to set the thresholds. The distance threshold was set to 0.002 degrees (or approximately 222 meters) so as to be small enough to distinguish two close but separate roads. The density threshold was set to the average number of points within the distance threshold to identify congested regions. For the other datasets, a heuristic scheme was adopted based on the K-distance graph used in the previous studies [15, 40]. The default settings of the density and distance thresholds as well as the window size for each dataset are summarized in Table 5.

Table 5: Threshold values and window sizes

Dataset	dim	density (τ)	distance (ϵ)	$\|window\|$
DTG	2D	372	0.002	2M (~10 min)
GeoLife	3D	765	0.002	0.1M (~ week)
IRIS	4D	8	2	0.2M (~ decade)
Household	7D	14	0.3	0.5M (~ year)

6.2 Baseline Evaluation

For the baseline performance evaluation, the update latency of each clustering method under the sliding window model is presented in Figure 7. For each dataset, the time taken to update clusters was measured when the window advanced by a single stride. The stride size was set to 5% of the window size, whose default settings are given in Table 5. The update latency is broken down to the insertion and deletion latency, and each measurement is the average of five runs. Since Extra-N does not support insertion and deletion operations separately, only a combined latency is shown in the figure.

DenForest and its non-optimized version (*DenForest-NO*) outperformed all the other methods. *DenForest* was up to 3.5 times faster than the second-best performer (DISC in the case of Household). IncDBSCAN yielded poor performance particularly for the GeoLife dataset. The reason is the GeoLife dataset is highly skewed in certain areas, which elongates the time taken for range searches significantly. DISC also relies on range searches but it is less affected by the skewedness of the dataset. This is because it takes advantage of optimized range searches such as epoch-based probes that reduce the redundant retrieval of data points.

Approx-Low and Approx-High showed poor performance for all the datasets. To determine whether a point is a *core* or not, the approximate method invokes a number of approximate range counting queries. Not only is it a major bottleneck but also it is aggravated as τ gets larger or as the number of dimensions increases. For the Household dataset, it did not even terminate within the allotted time of ten hours.

For all the datasets, the deletion latency of *DenForest* was much lower than the other methods. Except for the DTG dataset, the cost of deletion of *DenForest* was almost negligible. Figure 8 shows the speedup ratio of the deletion operation by *DenForest* when compared to the second-best performer in the log scale. The deletion time taken for processing a single stride was measured with the stride size set to 5% of the default window size. For the GeoLife dataset, the measurement of the third best performer (DISC) was used because the second best performer (Extra-N) does not support the insert and delete operations separately.

For a *vanishing* core, *DenForest* simply cuts the links incident to the *vanishing* core in $\mathcal{MST}$ to update the connectedness of the cluster, while DISC (second best performer) and IncDBSCAN invoke consecutive range searches in a BFS way. This contributes to the major performance improvement by *DenForest* over the other methods. Furthermore, it is less affected by the dimensionality, since the deletion by *DenForest* does not involve any range search. Consequently, the performance gap in the deletion operations increased with the increase of dimensionality. For the 7-dimensional Household dataset, *DenForest* achieved 56 times higher deletion speed than DISC.

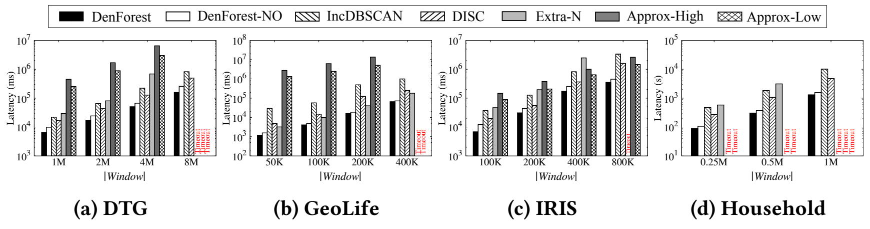
6.3 Varying Size of Window/Stride

The window size and the stride size can vary depending on the applications. Thus, the update latency was measured under various window sizes (Figure 9) and various stride sizes (Figure 10). The density (τ) and distance (ϵ) thresholds were set to the values in Table 5. Both *DenForest* and *DenForest-NO* outperformed the other clustering methods significantly with a wide margin for all the window sizes and for all the stride sizes. For some of the datasets, Extra-N and Approx-Low/High did not terminate with ten hours.

DenForest outperformed *DenForest-NO* across the entire spectrum of the window sizes and the stride sizes. The batch optimization of *DenForest* effectively lowered the cost of updating clusters by keeping the $\mathcal{MST}s$ smaller. On the other hand, the batch optimization incurs an additional overhead for managing *super nostalgic cores* and their neighbors. The amount of improvement by the batch optimization is also affected by the locality of data points in the same stride. The higher locality results in the more improvement. Therefore, increasing the stride size does not always contribute to performance gain by the batch optimization. On average, the batch optimization improved the performance about 25%.

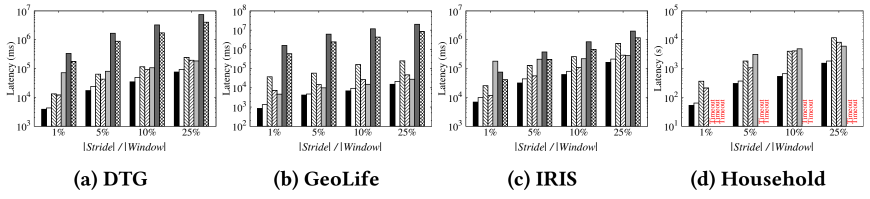
6.4 Effect of Density and Distance Thresholds

In this section, we used the DTG dataset to measure the effect of the density and distance thresholds on the performance. The insertion and deletion times taken to process a single stride were measured



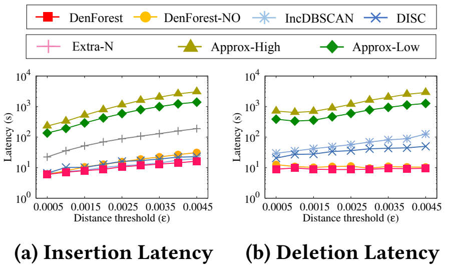
(a) DTG (b) GeoLife (c) IRIS (d) Household

Figure 9: Varying size of window (|Stride|/|Window|=5%)



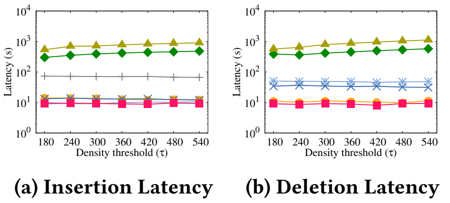
(a) DTG (b) GeoLife (c) IRIS (d) Household

Figure 10: Varying size of stride



(a) Insertion Latency (b) Deletion Latency

Figure 11: Varying ϵ for the DTG dataset



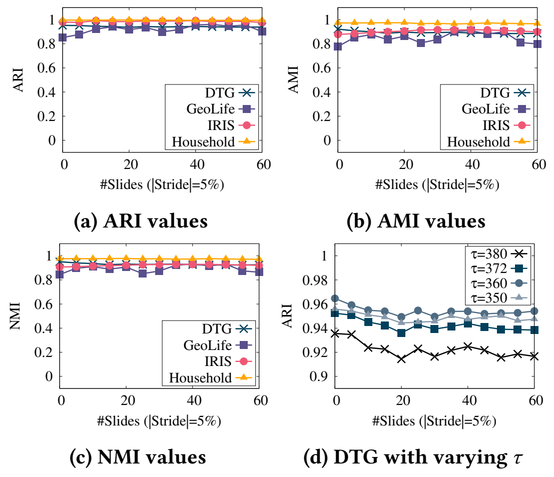
(a) Insertion Latency (b) Deletion Latency

Figure 12: Varying τ for the DTG dataset

for each clustering method. The window size was set to two million points, and the stride size was set to 5% of the window size.

Figure 11 shows the insertion and deletion latency with a varying distance threshold (ϵ). The density threshold was set to the default value in Table 5. The larger ϵ value generally requires the more time for range searches. Thus, the insertion and deletion latency increased as the ϵ threshold increased for all the clustering methods except for deletion by *DenForest* and *DenForest-NO*. The reason is of course they do not require any range search for deletion.

A similar experiment was conducted by varying the density threshold (τ) with the distance threshold fixed to the default value. Figure 12 shows that the density threshold hardly affected the performance except for Approx-High and Approx-Low, which slowed down as the density threshold increased. The reason is that the approximate method takes more time to determine whether a point is a core or not as the density threshold increases. A similar trend was also observed in the previous study [50].



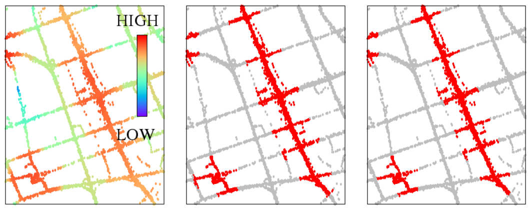
(a) ARI values (b) AMI values (c) NMI values (d) DTG with varying τ

Figure 13: Clustering quality over sliding windows

6.5 Clustering Quality over Sliding Windows

The clustering quality of *DenForest* was measured for each dataset. The true cluster labels are not available for the datasets. So we used the clustering results from DBSCAN as the ground truth. Three metrics ARI [27], AMI [49], and NMI [33] were used to measure the quality.

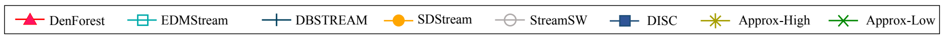
Figures 13a to 13c show how the clustering quality changes over time while the sliding window advances. The stride size was set to 5% of the window size. *DenForest* achieved clustering quality measurements close to one (or 100%) for all the datasets and sustained its quality as the window slid. The average measurements of quality were 0.96 (ARI), 0.91 (AMI) and 0.93 (NMI). We also observed the tendency that *DenForest* could improve its quality of clustering by choosing slightly lower density thresholds (τ) than those chosen for DBSCAN. This is shown in Figure 13d that measures the quality of clusters produced for the DTG dataset. *DenForest* and *DenForest-NO* produce the same clustering results. Thus, their quality measurements are identical.

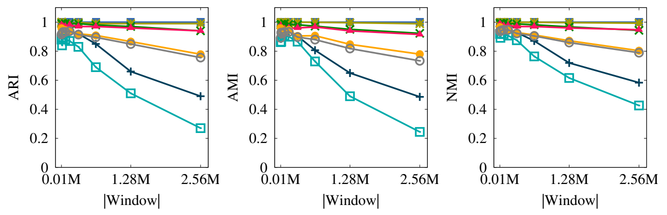


(a) Heatmap (b) DBSCAN (c) *DenForest*

Figure 14: Clusters found in DTG

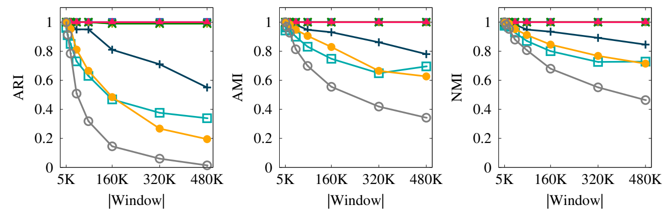
Figure 14 shows the heatmap and the examples of clusters detected by DBSCAN and *DenForest* for a snapshot (or window) of the DTG dataset. *DenForest* produced the result nearly identical to the result of DBSCAN, and *DenForest* detected dense areas matching well the heatmap that visualized the congested regions.





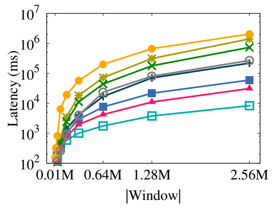
(a) ARI (b) AMI (c) NMI

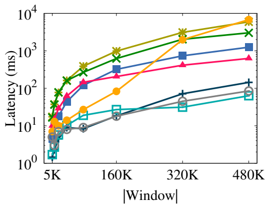
Figure 15: Quality of DTG clusters



(a) ARI (b) AMI (c) NMI

Figure 16: Quality of MAZE clusters





(a) DTG (b) MAZE

Figure 17: Latency with DTG and MAZE

6.6 Comparison with Summarization-Based Methods

DenForest was also compared with the summarization-based methods. DBSTREAM [25] is chosen because it is shown to achieve high quality in the previous study [5]. EDMStream [23] is a streaming version of the static density peak clustering algorithm [39]. SDStream [38] and StreamSW [43] are designed for the sliding window model based on *EHCF* [55] and grids [46], respectively. [2]

The *unlabeled real* DTG and the *labeled synthetic* Maze [31] datasets were used in the evaluation, and three metrics (ARI, AMI, and NMI) were applied to measure the quality with various window sizes. For the unlabeled DTG dataset, the clusters produced by DBSCAN were used as the ground truth. The parameters for EDMStream, DBSTREAM, SDStream, and StreamSW were tuned so as to achieve the highest quality for each window size. The update latency of one stride was also measured when the stride size was set to 5% of the window size. Only the insertion latency was included in the measurements for EDMStream and DBSTREAM, because they do not support a deletion operation.

The summarization-based methods assume an infinite length of data streams and summarize a group of data points into a *micro-cluster*. Since they only maintain coarse-grained information, quality of these methods decreased steeply as the window size increased as is shown in Figures 15 and 16. Although SDStream and StreamSW achieved relatively higher quality than other summarization methods for the DTG dataset, their quality was still lower than that of *DenForest.* Moreover, they were far slower than *DenForest* due to the high cost of maintaining a number of micro-clusters (in Figure 17). Conversely, *DenForest* attained high quality based on all the data points without approximation, and achieved the best performance among the methods whose quality was higher than 0.9.

7 RELATED WORKS

In addition to the incremental clustering algorithms such as Incremental DBSCAN [14], Extra-N [51], ρ-double-approximate DBSCAN [21] and DISC [31] described in Section 1, there are numerous summarization-based approaches taken to deal with density-based clustering over streaming data [4, 7, 23, 25, 37, 38, 43]. They assume a finite memory capacity for an infinite length of data streams. Thus, they maintain the summary of data points as *micro-clusters* instead of individual data points. These methods are good at discovering clusters quickly from the infinite data streams; they consume less memory and generally show low latency. However, they cannot capture the clusters accurately in real time and cannot achieve high clustering quality enough to replace the exact approaches such as DBSCAN.

It is also worth noting that parallelization of the DBSCAN algorithm has been studied actively in the past few years. NG-DBSCAN is one of the early work developed on the Spark framework as a scalable solution to density-based clustering [35]. RP-DBSCAN is a parallel DBSCAN algorithm that takes advantage of the random split strategy [45]. Wang *et al.* have proposed several exact and approximate DBSCAN algorithms based on grid construction and solving the bichromatic closest pairs problem in parallel [50].

8 CONCLUSION

This paper proposes a novel incremental density-based clustering algorithm called *DenForest* in order to address the slow deletion problem, which is inherent in the state-of-the-art clustering approaches. *DenForest* is based on a new notion of cores, namely *nostalgic cores*, proposed in this paper and achieves substantially higher performance by maintaining clusters as a group of *DenTrees* rather than a graph. The efficiency of *DenForest* is demonstrated by an extensive comparative evaluation conducted with the state-of-the-art clustering algorithms. Furthermore, it is observed that *DenForest* achieves high-quality clusters, comparable with that of DBSCAN, for numerous labeled synthetic and unlabeled real-world datasets. *DenForest* is expected to support many data analytic tasks in the streaming environment by clustering time-varying data efficiently at low computational cost.

[2]The Java code for EDMStream is available in https://github.com/ShufengGong/EDMStream. We implemented DBSTREAM, SDStream, and StreamSW in Java.

REFERENCES

[1] Nikos Armenatzoglou and Dimitris Papadias. 2018. Geo-Social Networks. In *Encyclopedia of Database Systems*, Ling Liu and M. Tamer Özsu (Eds.). Springer, New York, NY, 1620–1623.

[2] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM* 18, 9 (Sept. 1975), 509–517.

[3] Jon Louis Bentley. 1979. Decomposable searching problems. *Information Processing Letters* 8, 5 (1979), 244–251.

[4] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. 2006. Density-Based Clustering over an Evolving Data Stream with Noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*. Bethesda, MD, USA, 328–339.

[5] Matthias Carnein, Dennis Assenmacher, and Heike Trautmann. 2017. An Empirical Comparison of Stream Clustering Algorithms. In *Proceedings of the Computing Frontiers Conference*. Siena, Italy, 361–366.

[6] Hong Chang and Dit-Yan Yeung. 2008. Robust path-based spectral clustering. *Pattern Recognition* 41 (01 2008), 191–203.

[7] Yixin Chen and Li Tu. 2007. Density-Based Clustering for Real-time Stream Data. In *Proceedings of the 13th ACM SIGKDD Conference*. San Jose, California, USA, 133–142.

[8] Francis Chin and David Houck. 1978. Algorithms for Updating Minimal Spanning Trees. *Journal of Computer and System Sciences* 16, 3 (1978), 333–344.

[9] J. H. Conway and N. J. A. Sloane. 1999. Sphere Packings and Kissing Numbers. In *Sphere Packings, Lattices and Groups*. Springer, New York, NY, 1–30.

[10] T. Czerniawski, B. Sankaran, M. Nahangi, C. Haas, and F. Leite. 2018. 6D DBSCAN-based segmentation of building point clouds for planar object classification. *Automation in Construction* 88 (2018), 44 – 58.

[11] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining Stream Statistics over Sliding Windows (Extended Abstract). In *Proceedings of the 13th ACM-SIAM SODA Conference*. San Francisco, California, 635–644.

[12] DTG 2016. How to Use a Digital Tachograph. https://www.optac.info/uk/digital-tachograph/.

[13] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, and Xiaowei Xu. 1998. Incremental Clustering for Mining in a Data Warehousing Environment. In *Proceedings of the 24th VLDB Conference*. San Francisco, CA, USA, 323–333.

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd KDD Conference*. Portland, Oregon, 226–231.

[16] Tianhui Fan, Naijing Guo, and Yujie Ren. 2019. Consumer clusters detection with geo-tagged social network data using DBSCAN algorithm: a case study of the Pearl River Delta in China. *GeoJournal* 86 (09 2019), 317––337.

[17] Roberto Ferrara, Salvatore G.P. Virdis, Andrea Ventura, Tiziano Ghisu, Pierpaolo Duce, and Grazia Pellizzaro. 2018. An automated approach for wood-leaf separation from terrestrial LIDAR point clouds using the density based clustering algorithm DBSCAN. *Agricultural and Forest Meteorology* 262 (2018), 434 – 444.

[18] Pasi Fränti, Radu Mariescu-Istodor, and Caiming Zhong. 2016. XNN Graph. In *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshop*. Mérida, Mexico, 207–217.

[19] Joao Gama. 2010. *Knowledge Discovery from Data Streams* (1st ed.). Chapman & Hall/CRC, Porto, Portugal. 16–18 pages.

[20] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *Proceedings of the 2015 ACM SIGMOD Conference*. Melbourne, Victoria, Australia, 519–530.

[21] Junhao Gan and Yufei Tao. 2017. Dynamic Density Based Clustering. In *Proceedings of the 2017 ACM SIGMOD Conference*. Chicago, Illinois, USA, 1493–1507.

[22] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. 2007. Clustering Aggregation. *ACM TKDD* 1, 1 (March 2007), 1–30.

[23] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2017. Clustering Stream Data by Exploring the Evolution of Density Mountain. *Proc. VLDB Endow.* 11, 4 (2017), 393–405.

[24] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD Conference*. Boston, Massachusetts, USA, 47–57.

[25] M. Hahsler and M. Bolaños. 2016. Clustering Data Streams Based on Shared Density between Micro-Clusters. *IEEE TKDE* 28, 6 (2016), 1449–1461.

[26] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. 2001. Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity. *Journal of the ACM* 48, 4 (2001), 723–760.

[27] Lawrence Hubert and Phipps Arabie. 1985. Comparing Partitions. *Journal of Classification* 1 (1985), 193–218.

[28] IRIS 2022. Incorporated Research Institutions for Seismology. http://service.iris.edu/fdsnws/event/1/.

[29] Heinrich Jiang. 2017. Density Level Set Estimation on Manifolds with DBSCAN. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, Sydney, NSW, Australia, 1684–1693.

[30] Heinrich Jiang, Jennifer Jang, and Jakub Lacki. 2020. Faster DBSCAN via subsampled similarity queries. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 22407–22419.

[31] B. Kim, K. Koo, J. Kim, and B. Moon. 2021. DISC: Density-Based Incremental Clustering by Striding over Streaming Data. In *2021 IEEE 37th ICDE Conference*. 828–839.

[32] Junghoon Kim, Tao Guo, Kaiyu Feng, Gao Cong, Arijit Khan, and Farhana M. Choudhury. 2020. Densely Connected User Community and Location Cluster Search in Location-Based Social Networks. In *Proceedings of the 2020 ACM SIGMOD Conference*. Portland, OR, USA, 2199–2209.

[33] Tarald O. Kvalseth. 1987. Entropy and Correlation: Some Comments. *IEEE Transactions on Systems, Man, and Cybernetics* 17, 3 (1987), 517–519.

[34] Chung-Hong Lee. 2012. Mining Spatio-Temporal Information on Microblogging Streams Using a Density-Based Online Clustering Method. *Expert Systems with Applications* 39, 10 (Aug. 2012), 9623–9641.

[35] Alessandro Lulli, Matteo Dell'Amico, Pietro Michiardi, and Laura Ricci. 2016. NG-DBSCAN: Scalable Density-Based Clustering for Arbitrary Data. *Proc. VLDB Endow.* 10, 3 (Nov. 2016), 157–168.

[36] Nicholas Metropolis and S. Ulam. 1949. The Monte Carlo Method. *Journal of the American Statistical Association* 44, 247 (1949), 335–341.

[37] Irene Ntoutsi, Arthur Zimek, Themis Palpanas, Peer Kröger, and Hans-Peter Kriegel. 2012. Density-based Projected Clustering over High Dimensional Data Streams. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. München, Germany, 987–998.

[38] Jiadong Ren and R. Ma. 2009. Density-Based Data Streams Clustering over Sliding Windows. In *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Vol. 5. Tianjin, China, 248–252.

[39] Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *Science* 344, 6191 (2014), 1492–1496.

[40] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM TODS* 42, 3, Article 19 (July 2017), 21 pages.

[41] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao. 2016. Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm. *IEEE Transactions on Image Processing* 25, 12 (2016), 5933–5942.

[42] Yossi Shiloach and Shimon Even. 1981. An On-Line Edge-Deletion Problem. *J. ACM* 28, 1 (Jan. 1981), 1–4.

[43] K. Shyam Sunder Reddy and C. Shoba Bindu. 2019. StreamSW: A density-based approach for clustering data streams over sliding windows. *Measurement* 144 (2019), 14–19.

[44] Daniel D. Sleator and Robert Endre Tarjan. 1981. A Data Structure for Dynamic Trees. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, Milwaukee, Wisconsin, USA, 114–122.

[45] Hwanjun Song and Jae-Gil Lee. 2018. RP-DBSCAN: A Superfast Parallel DBSCAN Algorithm Based on Random Partitioning. In *Proceedings of the 2018 ACM SIGMOD Conference*. Houston, TX, USA, 1173–1187.

[46] Li Tu and Yixin Chen. 2009. Stream Data Clustering Based on Grid Density and Attraction. *ACM TKDD* 3, 3, Article 12 (July 2009), 27 pages.

[47] Cor Veenman, Marcel Reinders, and Eric Backer. 2002. A Maximum variance Cluster Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (10 2002), 1273– 1280.

[48] Suresh Venkatasubramanian. 2009. Clustering on Streams. In *Encyclopedia of Database Systems*, Ling Liu and M. Tamer Özsu (Eds.). Springer, New York, NY, 378–383.

[49] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2009. Information Theoretic Measures for Clusterings Comparison: Is a Correction for Chance Necessary?. In *Proceedings of the 26th ICML Conference*. Montreal, Quebec, Canada, 1073–1080.

[50] Yiqiu Wang, Yan Gu, and Julian Shun. 2020. Theoretically-Efficient and Practical Parallel DBSCAN. In *Proceedings of the 2020 ACM SIGMOD Conference*. Portland, OR, USA, 2555–2571.

[51] Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. 2009. Neighbor-Based Pattern Detection for Windows over Streaming Data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. Saint Petersburg, Russia, 529–540.

[52] Yang, Di and Rundensteiner, Elke A. and Ward, Matthew O. 2011. Summarization and Matching of Density-Based Clusters in Streaming Environments. *Proc. VLDB Endow.* 5, 2 (2011), 121–132.

[53] C. T. Zahn. 1971. Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters. *IEEE Transactions on Computers* C-20, 1 (1971), 68–86.

[54] Yu Zheng, Hao Fu, Xing Xie, Wei-Ying Ma, and Quannan Li. 2011. Geolife GPS trajectory dataset - User Guide. https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/.

[55] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. 2008. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems* 15 (05 2008), 181–214.