Dynamic DBSCAN with Euler Tour Sequences

Seiyun Shin[1] **Ilan Shomorony**[1] **Peter Macgregor**[2]
[1]University of Illinois Urbana-Champaign [2]University of St Andrews

Abstract

We propose a fast and dynamic algorithm for Density-Based Spatial Clustering of Applications with Noise (DBSCAN) that efficiently supports online updates. Traditional DBSCAN algorithms, designed for batch processing, become computationally expensive when applied to dynamic datasets, particularly in large-scale applications where data continuously evolves. To address this challenge, our algorithm leverages the Euler Tour Trees data structure, enabling dynamic clustering updates without the need to reprocess the entire dataset. This approach preserves a near-optimal accuracy in density estimation, as achieved by the state-of-the-art static DBSCAN method (Esfandiari et al., 2021). Our method achieves an improved time complexity of $O(d\log^3(n) + \log^4(n))$ for every data point insertion and deletion, where n and d denote the total number of updates and the data dimension, respectively. Empirical studies also demonstrate significant speedups over conventional DBSCANs in real-time clustering of dynamic datasets, while maintaining comparable or superior clustering quality.

1 INTRODUCTION

Density-based clustering is a fundamental problem in data science, with machine learning applications spanning computer vision (Shen et al., 2016), and medical imaging (Tran et al., 2012; Baselice et al., 2015). The key idea behind density-based clustering is to group points in space by identifying connected regions with high data density while separating them from sparser areas. One of the most widely used methods in this category is Density-Based Spatial Clustering of Applications with Noise (DBSCAN), introduced by Ester et al. (1996). DBSCAN estimates the density of each point by counting the number of points within its neighborhood and classifies those with densities above a threshold as core points. A neighborhood graph of core points is then constructed, with clusters formed based on the connected components. This method has been successfully adopted and integrated into numerous data mining tools (Hall et al., 2009; Pedregosa et al., 2011a; Team et al., 2013; Schubert et al., 2015).

Despite DBSCAN's success across many domains, it faces two major challenges: (1) its quadratic time complexity and (2) the requirement to process the entire dataset upfront. These limitations make it impractical for large-scale datasets that evolve *dynamically* over time. Static algorithms with polynomial-time complexity are no longer practical for such massive and evolving data.

To address the first challenge, several efficient and scalable algorithms have been proposed (Chen et al., 2005; Gunawan and de Berg, 2013; Gan and Tao, 2017; de Berg et al., 2019; Jang and Jiang, 2019). However, as the dimensionality d of the data increases, these algorithms still exhibit a running time similar to $O(n^2)$, where n denotes the number of data points. A notable improvement is DBSCAN++ (Jang and Jiang, 2019), which achieves faster clustering while maintaining the accuracy of the original DBSCAN under β-regularity assumption (to be specified later). Building on the work from Rinaldo and Wasserman (2010) that reveals the relationship between connected components of a neighborhood graph and the density level sets, DBSCAN++ has been shown to be near-optimal for λ-density level set estimation with respect to *Hausdorff distance*, with a time complexity of $O(n^{2-\frac{2\beta}{2\beta+d}})$. However, even this method struggles to avoid quadratic time complexity for high-dimensional datasets.

More recently, a near-linear time algorithm with $O(dn\log^2(n))$ complexity was proposed by Esfandiari et al. (2021) which makes use of locality sensitive hash funcions, and achieves a near-optimal Hausdorff

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258.

distance for density level sets. While this method offers significant improvements due to the decoupled dependency on the number of data points n and the data dimension d in its runtime guarantee, the algorithm remains designed primarily for static, batch-processing tasks. Consequently it becomes computationally expensive for dynamic datasets where data points are continuously added or removed. In particular, using this static algorithm for even a single point update requires reprocessing the entire dataset, meaning it incurs the same time complexity as clustering all n data points, making it inefficient for large-scale applications.

Our work addresses both the computational and dynamic challenges by introducing a fast, dynamic DBSCAN algorithm that efficiently supports online updates. By leveraging the *Euler Tour Sequence* data structure for dynamic forests (Henzinger and King, 1995; Tseng et al., 2019), our algorithm enables dynamic clustering updates without the need to reprocess all data points, thereby significantly reducing computational overhead. This approach preserves a near-optimal accuracy in density estimation as achieved by the state-of-the-art static DBSCAN method (Esfandiari et al., 2021), with a time complexity of $O(d\log^3(n) + \log^4(n))$ for every data point insertion and deletion. Here n denotes the maximum number of data points at any time.

Through extensive empirical studies, we demonstrate that our algorithm provides substantial speed improvements over the state-of-the-art static DBSCAN, as well as the conventional DBSCAN method, particularly for real-time clustering in dynamic datasets. Not only does it deliver superior computational efficiency, but it also maintains or improves clustering quality.

2 PRELIMINARIES

Consider a dynamic setting where the dataset $X \subseteq \mathbb{R}^d$ evolves over time as new data points are added or existing ones are removed. We assume that each data point is drawn independently and identically distributed (i.i.d.) from a distribution $\mathcal{F}$ over $\mathbb{R}^d$. The goal is to *dynamically* partition the evolving dataset into distinct clusters based on the density of the data points.

2.1 DBSCAN

The foundational DBSCAN algorithm (Ester et al., 1996) builds on the idea of identifying high-density regions in a static dataset to form clusters. The algorithm first estimates points that belong to density level set, where points in dense regions are referred to as core points. These core points serve as an approximation of the density level set. The algorithm

Algorithm 1 DBSCAN(k, ε, X)

Initialise set of core points $C = \emptyset$
Initialise empty graph G
for $\boldsymbol{x} \in X$ **do**
    $C \leftarrow C \cup \{\boldsymbol{x}\}$, if $|\{\boldsymbol{y} \in X : \mathsf{dist}(\boldsymbol{x}, \boldsymbol{y}) \leq \varepsilon\}| \geq k$
Construct a graph $G = (V, E)$, where $V \leftarrow X$
for $\boldsymbol{c} \in C$ **do**
    Add edge $e = (\boldsymbol{c}, \boldsymbol{x}) \in E$, $\forall \boldsymbol{x} \in X \cap \mathcal{B}(\boldsymbol{c}, \varepsilon)$
Return connected components of G

then identifies connected components of these dense regions by constructing a graph where core points within distance ε are connected. The final clusters are the connected components of this graph, with non-core points assigned to nearby dense regions and remaining points treated as noise or outliers. According to Algorithm 1, a point $\boldsymbol{x}$ is classified as a core point if there are at least k points within its ε-radius neighborhood $\mathcal{B}(\boldsymbol{x}, \varepsilon)$ The neighborhood is determined by a distance metric, which is defined in Euclidean space as follows:

Definition 1. *Given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$, and a set $C \subseteq \mathbb{R}^d$, define $\mathsf{dist}(\boldsymbol{x}, \boldsymbol{y}) := \|\boldsymbol{x} - \boldsymbol{y}\|_2 = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$; also define a distance between $\boldsymbol{x}$ and C to be $\mathsf{dist}(\boldsymbol{x}, C) := \inf_{\boldsymbol{y} \in C} \|\boldsymbol{x} - \boldsymbol{y}\|_2$. For $\varepsilon > 0$, define $\mathcal{B}(\boldsymbol{x}, \varepsilon) := \{\boldsymbol{y} \in \mathcal{X} : \mathsf{dist}(\boldsymbol{x}, \boldsymbol{y}) \leq \varepsilon\}$ and $\mathcal{B}(C, \varepsilon) := \{\boldsymbol{y} \in \mathcal{X} : \mathsf{dist}(\boldsymbol{y}, C) \leq \varepsilon\}$, respectively.*

To establish theoretical guarantees, subsequent works (Jiang, 2017; Jang and Jiang, 2019) assume the existence of an underlying, yet *unknown*, density function $f : \mathcal{X} \to \mathbb{R}_{\geq 0}$, where $\mathcal{X} \subseteq \mathbb{R}^d$ denotes the support of $\mathcal{F}$. The density of any given data point $\boldsymbol{x}$ is then defined as $f(\boldsymbol{x})$. In particular, λ-density level set (or simply λ-level set) is defined as:

Definition 2 (Density Level Set)**.** *We define the λ-level set of f as: $L_f(\lambda) := \{\boldsymbol{x} \in \mathcal{X} : f(\boldsymbol{x}) \geq \lambda\}$. Here λ characterises the density level.*

The algorithm is given λ as input, but does not have access to f. Building on the work from Rinaldo and Wasserman (2010), which demonstrates that partitioning via a neighborhood graph can effectively approximate the clustering of level sets, Jiang (2017); Jang and Jiang (2019) provide comprehensive studies on how to choose (k, ε) to estimate the density level set for a given threshold λ, and provide bounds on the Hausdorff distance error. In practical situations, where λ is not explicitly provided, (k, ε) are treated as hyperparameters to optimise the clustering performance.

Following this framework, our proposed algorithm will also use (k, ε) as hyperparameters, with theoretical guarantees based on the λ-level set. The relationship between (k, ε) and the density level λ will be explicitly

discussed in Section 4.

2.2 Dynamic Forests

A dynamic forest is a data structure which provides the following operations:

- ADD($\boldsymbol{x}$): add a new node to the forest.
- LINK($\boldsymbol{x}, \boldsymbol{y}$): connect $\boldsymbol{x}$ and $\boldsymbol{y}$, if they are in different trees.
- CUT($\boldsymbol{x}, \boldsymbol{y}$): remove any edge between $\boldsymbol{x}$ and $\boldsymbol{y}$.
- ROOT($\boldsymbol{x}$): return the root of the tree having $\boldsymbol{x}$.

Henzinger and King (1995) proposed the use of Euler Tours Trees in order to implement this data structure, with a running time of $O(\log(n))$ for the LINK, CUT, and ROOT operations (also we refer to Demaine (2012)). Instead of storing the represented forest directly, this approach stores the Euler Tour Sequence of each tree in the forest. By viewing the tree as a directed graph, where each edge is represented by two directed edges, the Euler Tour Sequence is the sequence of nodes visited when performing an Euler tour of the tree beginning with the root.

Henzinger and King (1995) demonstrated that the dynamic forest operations can be efficiently implemented by storing the Euler Tour Sequences in a balanced binary tree data structure, with appropriate joining and splitting when edges are added or removed from the forest.

More recently, Tseng et al. (2019) proposed another scheme based on the same idea, in which the Euler Tour Sequences are instead represented using a skip list data structure. This approach preserves the same asymptotic guarantees while simplifying the implementation considerably, and it is this technique that we employ for our proposed DBSCAN algorithm.

3 DYNAMIC DBSCAN ALGORITHM

Our algorithm builds on the nearly-linear time algorithm for DBSCAN given by Esfandiari et al. (2021). In contrast to the standard DBSCAN described in Algorithm 1, they observe that locality-sensitive hash functions can be used to efficiently find close data points without an expensive linear scan of the dataset. We will use hash functions defined as follows:

Definition 3 (Hash Function)**.** *For a point $\boldsymbol{x} \in \mathbb{R}^d$ and parameter ε, we define a hash function $h(\boldsymbol{x}) := \lfloor \frac{\boldsymbol{x}+\eta \cdot \mathbb{1}_d}{2\varepsilon} \rfloor$, where the floor function is applied entry-wise to the vector, η is drawn independently from the uniform distribution over $[0, 2\varepsilon]$, and $\mathbb{1}_d$ is the d-dimensional all-ones vector.*

Algorithm 2 DYNAMICDBSCAN

procedure INITIALISE(k, t, ε)
  Initialise hash functions $\{h_i\}_{i=1}^t$, dynamic forest G, and set of core points $C = \emptyset$.
procedure ADDPOINT($\boldsymbol{x}$)
  G.ADD($\boldsymbol{x}$)
  $C' = \emptyset$ ▷ New core points
  for $i \in [t]$ **do**
    Compute $h_i(\boldsymbol{x})$ and add $\boldsymbol{x}$ to $\mathsf{bucket}_i(\boldsymbol{x})$
    if $|\mathsf{bucket}_i(\boldsymbol{x})| > k$ **then**
      $C' \leftarrow C' \cup \{\boldsymbol{x}\}$
    else if $|\mathsf{bucket}_i(\boldsymbol{x})| = k$ **then**
      $C' \leftarrow C' \cup (\mathsf{bucket}_i(\boldsymbol{x}) \setminus C)$
  $C \leftarrow C \cup C'$
  for $\boldsymbol{c} \in C'$ **do**
    LINKCOREPOINT($\boldsymbol{c}$)
  if $C' = \emptyset$ **then**
    LINKNONCOREPOINT($\boldsymbol{x}$)
procedure DELETEPOINT($\boldsymbol{x}$)
  if $\boldsymbol{x}$ is a core point **then**
    $C' = \emptyset$ ▷ New non-core points
    for $i \in [t]$ **do**
      if $|\mathsf{bucket}_i(\boldsymbol{x})| = k$ **then**
        Add each $\boldsymbol{y} \in \mathsf{bucket}_i(\boldsymbol{x})$ to C' if $|\mathsf{bucket}_j(\boldsymbol{y})| < k$ for all $j \neq i$.
    $C \leftarrow C \setminus C'$
    for $\boldsymbol{c} \in C'$ **do**
      UNLINKCOREPOINT($\boldsymbol{c}$)
      LINKNONCOREPOINT($\boldsymbol{c}$)
  Remove $\boldsymbol{x}$ from G, C, and all hash tables.
procedure LINKCOREPOINT($\boldsymbol{x}$)
  Cut any edge incident to $\boldsymbol{c}$ in G.
  for $i \in [t]$ **do**
    $\boldsymbol{c}_1 \leftarrow \max\{\boldsymbol{c} \in \mathsf{bucket}_i(\boldsymbol{x}) : \mathsf{idx}(\boldsymbol{c}) < \mathsf{idx}(\boldsymbol{x})\}$
    $\boldsymbol{c}_2 \leftarrow \min\{\boldsymbol{c} \in \mathsf{bucket}_i(\boldsymbol{x}) : \mathsf{idx}(\boldsymbol{c}) > \mathsf{idx}(\boldsymbol{x})\}$
  Here $\max(\cdot)$ and $\min(\cdot)$ refer to returning the point with the highest and lowest index, respectively.
    G.CUT($\boldsymbol{c}_1, \boldsymbol{c}_2$)
    G.LINK($\boldsymbol{c}_1, \boldsymbol{x}$)
    G.LINK($\boldsymbol{x}, \boldsymbol{c}_2$)
procedure UNLINKCOREPOINT($\boldsymbol{x}$)
  for $i \in [t]$ **do**
    $\boldsymbol{c}_1 \leftarrow \max\{\boldsymbol{c} \in \mathsf{bucket}_i(\boldsymbol{x}) : \mathsf{idx}(\boldsymbol{c}) < \mathsf{idx}(\boldsymbol{x})\}$
    $\boldsymbol{c}_2 \leftarrow \min\{\boldsymbol{c} \in \mathsf{bucket}_i(\boldsymbol{x}) : \mathsf{idx}(\boldsymbol{c}) > \mathsf{idx}(\boldsymbol{x})\}$
    G.CUT($\boldsymbol{c}_1, \boldsymbol{x}$)
    G.CUT($\boldsymbol{x}, \boldsymbol{c}_2$)
    G.LINK($\boldsymbol{c}_1, \boldsymbol{c}_2$)
  Re-link any non-core points attached to $\boldsymbol{x}$.
procedure LINKNONCOREPOINT($\boldsymbol{x}$)
  Link $\boldsymbol{x}$ with **one** core point in $\mathsf{bucket}_i(\boldsymbol{x})$, for some i.
procedure GETCLUSTER($\boldsymbol{x}$)
  Return G.ROOT($\boldsymbol{x}$)

With this definition, Esfandiari et al. (2021) give the following guarantee for the recovery of close points.

Lemma 1 ((Esfandiari et al., 2021)). *Given $\varepsilon > 0$, the following holds for any two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and for a hash function h:*

1. $\Pr[h(\boldsymbol{x}) = h(\boldsymbol{y})] \geq 1 - \frac{\|\boldsymbol{x}-\boldsymbol{y}\|_1}{2\varepsilon}$;
2. $h(\boldsymbol{x}) = h(\boldsymbol{y}) \implies \|\boldsymbol{x} - \boldsymbol{y}\|_\infty \leq 2\varepsilon$.

Using this fact that close points are likely to have the same hash value, we create t independent hash functions $h_1, \ldots h_t$ and define the set of core points C for our DBSCAN algorithm as follows.

Definition 4 (Core Points). *For a dataset $X := \{\boldsymbol{x}_i\}_{i=1}^n$, the set of core points $C \subseteq X$ is*

$$C \triangleq \{\boldsymbol{x}_i \in X : \exists j \in [t] \text{ such that } |\mathsf{bucket}_j(\boldsymbol{x}_i)| \geq k\},$$

where $\mathsf{bucket}_j(\boldsymbol{x}_i) := \{\boldsymbol{x}_\ell \in X : h_j(\boldsymbol{x}_\ell) = h_j(\boldsymbol{x}_i)\}$.

We then construct a graph $H = (V, E)$, where each vertex $\boldsymbol{v}_i$ corresponds to the data point $\boldsymbol{x}_i$, and an edge $(\boldsymbol{v}_i, \boldsymbol{v}_j) \in E$ exists between any pair of core points $\boldsymbol{x}_i, \boldsymbol{x}_j \in C$, if $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ collide under any of the hash functions $h_1, \ldots h_t$. Esfandiari et al. (2021) show that with appropriate choices of the parameters ε, t, and k, the connected components of the graph H correspond to the connected components of the λ-density level set of the data.

For our dynamic application, however, maintaining the connected components of H as points are added and removed from the dataset is quite challenging. This difficulty arises primarily due to the fact that H is a dense graph with no inherent structure. In order to overcome this, we will maintain a *spanning forest* of H (as illustrated in Figure 1) throughout the dynamic updates, and guarantee that the maximum degree of any vertex in the spanning forest is at most $O(\log(n))$. By using an Euler tour sequence data structure to maintain the dynamic spanning forest, the update time of our dynamic data structure is bounded by $O(\log^2(n))$.

Our DynamicDBSCAN data structure is formally described in Algorithm 2. In each of the AddPoint($\boldsymbol{x}$) and RemovePoint($\boldsymbol{x}$) methods, we first compute the change to the set of core points resulting from adding (resp. removing) $\boldsymbol{x}$. Then, we update the connections for any core points added (resp. removed). Within each hash bucket, we always maintain that the core points are connected in a path structure according to the indices of the data points. This ensures that the degree of every core point in the spanning forest is bounded by $O(t)$, where t is the number of hash functions, and we set it to be $O(\log(n))$. Non-core points are each connected to at most one core point with which they collide in any hash function, ensuring that each non-core point has a degree of at most 1 in the spanning forest.

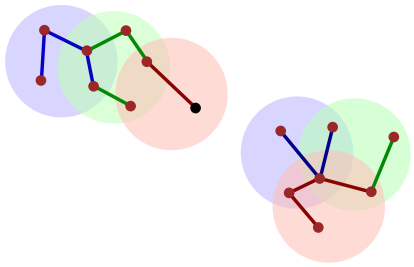


Figure 1: Illustration of a graph constructed by the DynamicDBSCAN algorithm: Red points represent core points, and each shaded region corresponds to a separate hash bucket. Edge colors match the hash bucket they belong to. Within each hash bucket, a path is added on the core points unless adding an edge would introduce a cycle into the graph.

Finally, our data structure always supports a GetCluster($\boldsymbol{x}$) query method, which returns a unique identifier of the cluster containing $\boldsymbol{x}$. This corresponds to returning the identifier of the tree containing $\boldsymbol{x}$ in the spanning forest. This can be obtained through a call to the root operation of the dynamic forest data structure, with a time complexity of $O(\log(n))$.

4 THEORETICAL RESULTS

4.1 Time Complexity

Theorem 1. *Let $X \subseteq \mathbb{R}^d$ be a set of n data points updated through data point insertions and deletions. The running time of the procedures in Algorithm 2 is as follows:*

1. Initialise(k, t, ε) *runs in $O(td)$ time.*
2. AddPoint($\boldsymbol{x}$) *and* DeletePoint($\boldsymbol{x}$) *run in $O(t^2k(d + \log(n))$ time.*
3. GetCluster($\boldsymbol{x}$) *runs in $O(\log(n))$ time.*

By setting $t = O(\log(n))$ and $k = O(\log(n))$, our data structure has update time $O(d\log^3(n) + \log^4(n))$.

Proof. The running time of the Initialise procedure is dominated by the initialisation of the t hash functions, each of which requires $O(d)$ time to create. Hence, the overall time complexity for Initialise is $O(td)$.

To establish the second claim, we first show that the running time of the LinkCorePoint, UnlinkCorePoint and LinkNonCorePoint operations are all bounded by $O(t \cdot (d + \log(n)))$. These follow from the two facts: (1) computing the hash values for $\boldsymbol{x}$ takes $O(td)$ time; and (2) the remaining operations, such as the cut and link operations, are efficiently handled

using the Euler Tour Sequence data structure, such that each operation takes $O(\log(n))$ time per hash functions, and searching for core points $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ within the hash buckets is optimised using a balanced binary tree. Furthermore, observe that in the AddPoint and DeletePoint methods, at most $O(tk)$ calls are made, as the number of affected core points $|C'|$ is bounded by $t \cdot k$. Consequently, these methods make at most $O(tk)$ calls to the LinkCorePoint, UnlinkCorePoint, and LinkNonCorePoint operations, resulting in a total running time of $O(t^2 \cdot k \cdot (d + \log(n)))$.

Finally, the GetCluster method involves a single call to the root operation on the Euler tour dynamic forest, having the time complexity of $O(\log(n))$. ■

Remark 1 (Comparison to Esfandiari et al. (2021))**.** *By leveraging the Euler tour sequence data structure, our algorithm efficiently handles dynamic updates without needing to reprocess the entire dataset, achieving a time complexity of* $O(d\log^3(n) + \log^4(n))$.

In contrast, the use of the the static algorithm proposed by Esfandiari et al. (2021) requires reconfiguration of the core and non-core point sets due to the update, as well as graph reconstruction. This process is essentially equivalent to re-running the algorithm from scratch with $n + 1$ *data points, resulting in a time complexity of* $O(tdn\log(n)) = O(dn\log^2(n))$ *when* $t = O(\log(n))$.

Remark 2 (Memory Complexity)**.** *The total memory usage of Algorithm 2 is* $O((n + d)\log(n))$. *This is due to the fact that (1) we use* $t = O(\log(n))$ *hash functions, requiring* $O(d\log(n))$ *space to store the hash functions themselves; (2)* $O(n\log(n))$ *space is required to store the data in the hash buckets; and lastly (3) we note that the space to store the spanning forest is* $O(n)$.

4.2 Correctness of Dynamic Updates

In this subsection, we prove that after each call to AddPoint or DeletePoint, our algorithm consistently correctly maintains that the subgraph of G, corresponding to the core points, as a spanning forest $H = (V_H, E_H)$. In this forest, each vertex $\boldsymbol{v}_i$ corresponds to a core point $\boldsymbol{x}_i$, and $(\boldsymbol{v}_i, \boldsymbol{v}_j) \in E_H$ if and only if there exists some $\ell \in [t]$ such that $h_\ell(\boldsymbol{x}_i) = h_\ell(\boldsymbol{x}_j)$.

Theorem 2. *After every call to* AddPoint *or* DeletePoint, $G[C]$ *(i.e., the subgraph of* G *induced by the core points) remains a spanning forest of* H.

Notice that since $G[C]$ is a spanning forest of H, and non-core points in G have degree at most 1, this immediately implies that the connected components of G correspond directly to the connected components of H. Also, since H is invariant to the order in which points are added and removed, and the clustering behaviour of DynamicDBSCAN is based only on connected components, it suffices to analyse H in order to obtain theoretical guarantees on the clustering quality.

Proof of Theorem 2. We proceed by induction. Assume that prior to any call to AddPoint or DeletePoint, $G[C]$ is a spanning forest of H. Then one can readily see that when AddPoint($\boldsymbol{x}$) (resp. DeletePoint($\boldsymbol{x}$)) is called, the algorithm correctly computes the set C' which represents the change in the core set induced by adding (resp. removing) the point $\boldsymbol{x}$.

Let $b_{i,j} := \{\boldsymbol{x} \in X : h_i(\boldsymbol{x}) = j\}$ be a hash bucket of h_i. By the inductive hypothesis that $G[C]$ is a spanning forest of H prior to the update, we see that every core point in $b_{i,j}$ is in the same connected component of G.

For AddPoint, Algorithm 2 calls LinkCorePoint for each new core point in C'. Following each call to LinkCorePoint, the algorithm maintains that every core point in $b_{i,j}$ is in the same connected component of G: notice that in each bucket, a link between $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ may be cut, but linking $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ again with the new point $\boldsymbol{x}$ ensures that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ remain in the same connected component.

The case for RemovePoint is similar: the algorithm calls UnlinkCorePoint($\boldsymbol{x}$) for each point removed from the set of core points. Although the links $(\boldsymbol{c}_1, \boldsymbol{x})$ and $(\boldsymbol{x}, \boldsymbol{c}_2)$ are cut, the algorithm ensures that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ remain connected by calling link($\boldsymbol{c}_1, \boldsymbol{c}_2$).

We highlight that the link($\boldsymbol{x}, \boldsymbol{y}$) operation of the dynamic forest data structure adds an edge between $\boldsymbol{x}$ and $\boldsymbol{y}$ only if they are not already in the same tree. This ensures that the structure of G remains a forest throughout the execution of the algorithm.

Finally, since edges are only added between core points that collide in some hash bucket, $G[C]$ is always a subgraph of H. Given that:

- $G[C]$ is a subgraph of H;
- every pair of points connected by an edge in H are in the same connected component of $G[C]$; and
- $G[C]$ is a forest;

we can conclude that $G[C]$ is a spanning forest of H after every call to AddPoint or RemovePoint. ■

4.3 Approximation of Density Level Sets

We are now ready to state the theoretical results for our algorithm with respect to finding density level sets. Notably, the near-linear time DBSCAN algorithm proposed by Esfandiari et al. (2021) has been shown to achieve a near-optimal statistical guarantee for estimating density level sets. In contrast to their algorithm, DynamicDBSCAN defines core points based on all

t hash functions, rather than using a dedicated hash function specifically for core point determination. This new definition ensures that no hash bucket contains more than k non-core points, a crucial requirement to guarantee poly-logarithmic update time for the data structure.

Armed with this, we will show that despite this modification, Algorithm 2 still dynamically maintains an estimator for the λ-density level set, achieving a near-optimal performance with respect to the Hausdorff distance.

4.3.1 Assumptions and Parameter Setup

Throughout the analysis, we follow prior work (Jiang, 2017; Jang and Jiang, 2019; Esfandiari et al., 2021) and make two regularity assumptions on the density distribution of the data.

Assumption 1. *f is continuous and has convex compact support $\mathcal{X} \subseteq \mathbb{R}^d$.*

Assumption 2 (β-regularity of level-sets)**.** *There exist $\beta, R_1, R_2, \lambda_c > 0$ such that $\forall \boldsymbol{x} \in L_f(\lambda - \lambda_c) \setminus L_f(\lambda)$, $R_1 \cdot \mathsf{dist}(\boldsymbol{x}, L_f(\lambda))^\beta \leq \lambda - f(\boldsymbol{x}) \leq R_2 \cdot \mathsf{dist}(\boldsymbol{x}, L_f(\lambda))^\beta$.*

The first assumption ensures that the distribution is continuous, which is a natural requirement. The second assumption, referred to as β-regularity, is a standard condition in level set analysis that characterises the prominence of the level set boundary. This is captured by β and λ_c.

For given values of n, d, β, λ, and λ_c corresponding to our target dataset, we choose parameters of the DynamicDBSCAN algorithm to satisfy

- $k \geq M_1 \cdot \left(\frac{\lambda}{\lambda_c}\right)^2 \cdot \omega(\log(n))$;
- $k \leq M_2 \cdot \lambda^{\frac{2\beta+2d}{2\beta+d}} \cdot O\left((\log(n))^{\frac{d}{2\beta+d}} \cdot n^{\frac{2\beta}{2\beta+d}}\right)$;
- $\varepsilon := \frac{1}{2}\left(\frac{k}{n\lambda(1-2C_{\delta,n}/\sqrt{k})}\right)^{\frac{1}{d}}$;
- $t := 100 \log\left(\frac{n}{\delta}\right)$;

where $C_{\delta,n} := C_0\left(\log(t/\delta)\sqrt{d\log(n)}\right)$ and δ is a confidence parameter, ensuring that our guarantees hold with a probability of at least $1-\delta$. Moreover, we consider the regime where n is sufficiently large with M_1 chosen to be sufficiently large and M_2 sufficiently small, both depending on d, and the density function $f(\cdot)$. While we treat the dimension d as a constant here, our results also hold for $d = \Theta(\log^c(n))$ for any constant $c > 0$. Additional details can be found in the Appendix.

The following uniform convergence bound from Chaudhuri and Dasgupta (2010) will serve as a key component in our analysis.

Lemma 2 (Lemma 7 of Chaudhuri and Dasgupta (2010))**.** *Let X be a set of n i.i.d. samples drawn from a distribution $\mathcal{F}$ over $\mathcal{X}$. With probability at least $1 - \frac{\delta}{t}$, the following conditions hold for any cube $K \subseteq \mathbb{R}^d$:*

1. *If $\Pr_{\boldsymbol{x}\sim\mathcal{F}}[\boldsymbol{x} \in K] \geq \frac{k}{n} + C_{\delta,n}\frac{\sqrt{k}}{n}$, then $|X \cap K| \geq k$.*
2. *If $\Pr_{\boldsymbol{x}\sim\mathcal{F}}[\boldsymbol{x} \in K] < \frac{k}{n} - C_{\delta,n}\frac{\sqrt{k}}{n}$, then $|X \cap K| < k$.*
3. *If $\Pr_{\boldsymbol{x}\sim\mathcal{F}}[\boldsymbol{x} \in K] \geq C_{\delta,n}\frac{\sqrt{d\log n}}{n}$, then $|X \cap K| \geq 1$.*

4.3.2 Density Level Set Estimation

We now establish that the connected components generated by our algorithm provide an accurate estimation of the target density level set. To this end, we first demonstrate that any sampled point located too far from the desired density level set will not be included in the core point set. Furthermore, we show that for any point within the desired density level set, there exists a nearby point that will be included in the core set. We note that this analysis follows that of Esfandiari et al. (2021) quite closely although we must carefully take account of the difference in our definition of core points. These findings are formalised in the following two lemmas.

Lemma 3. *For any point $\boldsymbol{x} \in X$, if $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \geq 2\left(\frac{\lambda}{R_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$, then $\boldsymbol{x}$ will not be added to the core point set C, with probability at least $1-\delta$.*

Proof. Let $\boldsymbol{x} \in X$ be such that $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \geq 2\left(\frac{\lambda}{R_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$ and consider any point $\boldsymbol{y} \in X$ satisfying $h_i(\boldsymbol{y}) = h_i(\boldsymbol{x})$ for some $i \in [t]$. By the second claim in Lemma 1, $\|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \sqrt{d}\|\boldsymbol{x} - \boldsymbol{y}\|_\infty \leq 2\sqrt{d}\varepsilon$. Hence, the distance from $\boldsymbol{y}$ to the level set $L_f(\lambda)$ can be bounded as follows: $\mathsf{dist}(\boldsymbol{y}, L_f(\lambda)) \geq \mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) - 2\sqrt{d}\varepsilon$. Since $\frac{C_{\delta,n}}{\sqrt{k}} = \left(\frac{1}{M_2}\right)^{\frac{1}{2\beta}} \cdot \omega\left((\log(n)/n)^{\frac{1}{2\beta+d}}\right)$ and $\varepsilon = M_2^{\frac{1}{d}} \cdot O\left((\log(n)/n)^{\frac{1}{2\beta+d}}\right)$, (detailed derivations are provided in Appendix B), it follows that $\mathsf{dist}(\boldsymbol{y}, L_f(\lambda)) \geq \frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$ for sufficiently small M_2. By Assumption 2, we have $f(\boldsymbol{y}) \leq \lambda - R_1 \cdot \mathsf{dist}(\boldsymbol{y}, L_f(\lambda))^\beta \leq \lambda - R_1 \cdot \left(\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}\right)^\beta \leq \lambda - 10\lambda\frac{C_{\delta,n}}{\sqrt{k}}$.

Now consider the condition for $\boldsymbol{x}$ to be added to the core set. For $\boldsymbol{x}$ to be a core point, there must be at least k points within its hash bucket; however, as $\Pr[\boldsymbol{x} \in K] = \int_{\mathcal{X}} f(\boldsymbol{y}) \cdot \mathbb{1}_{[h_i(\boldsymbol{y}) = h_i(\boldsymbol{x})]} d\boldsymbol{y} \leq (2\varepsilon)^d \left(\lambda - 10\lambda\frac{C_{\delta,n}}{\sqrt{k}}\right) \leq k \cdot \frac{\lambda - 10\lambda C_{\delta,n}/\sqrt{k}}{n\lambda(1-2C_{\delta,n}/\sqrt{k})} \leq \frac{k}{n} - 8C_{\delta,n}\frac{\sqrt{k}}{n}$, by Lemma 2, with probability at least $1 - \frac{\delta}{t}$, $|X \cap K| < k$. Applying the union bound for all $\{h_i\}_{i=1}^t$, we conclude that $\boldsymbol{x}$ is not

Name	n	d	Clusters	Reference
Letter	20000	16	26	Frey and Slate (1991)
MNIST	70000	20	10	Lecun et al. (1998)
Fashion-MNIST	70000	20	10	Xiao et al. (2017)
Blobs	200000	10	10	Synthetic Data
KDDCup99	494000	20	23	Stolfo et al. (1999)
Covertype	581012	54	7	Blackard and Dean (1999)

Table 1: Dataset Information

included in C, with probability at least $1-\delta$. ■

Lemma 4. *With probability at least $1-\delta$, any point $\boldsymbol{x} \in X$ satisfying $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \leq \frac{1}{2}\left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$ will be added to C.*

Proof. Let $\boldsymbol{x} \in X$ such that $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \leq \frac{1}{2}\left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$ and consider any point $\boldsymbol{y} \in X$ satisfying $h_i(\boldsymbol{y}) = h_i(\boldsymbol{x})$ for some $i \in [t]$. By Lemma 1, we have $\|\boldsymbol{x}-\boldsymbol{y}\|_2 \leq 2\sqrt{d}\varepsilon$. Hence, $\mathsf{dist}(\boldsymbol{y}, L_f(\lambda)) \leq \mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) + 2\sqrt{d}\varepsilon \leq \left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$, where the last inequality follows from the same order-wise comparison provided in the proof of Lemma 3. Assumption 2 then implies $f(\boldsymbol{y}) \geq \lambda - \lambda\frac{C_{\delta,n}}{\sqrt{k}}$, from which we have $\Pr[\boldsymbol{x} \in K] = \int_{\mathcal{X}} f(\boldsymbol{y})\cdot \mathbb{1}_{[h_i(\boldsymbol{y})=h_i(\boldsymbol{x})]}d\boldsymbol{y} \geq (2\varepsilon)^d\left(\lambda - \lambda\frac{C_{\delta,n}}{\sqrt{k}}\right) \geq k\cdot\frac{\lambda-\lambda C_{\delta,n}/\sqrt{k}}{n\lambda(1-2C_{\delta,n}/\sqrt{k})} \geq \frac{k}{n} + C_{\delta,n}\frac{\sqrt{k}}{n}$, implying from Lemma 2 that $|X\cap K| \geq k$, with probability at least $1-\delta$. Hence, $\boldsymbol{x}$ will be added to the core point set C. ■

Lemma 5. *For any point $\boldsymbol{x} \in L_f(\lambda)$, with probability at least $1-\delta$, there exists a core point $\boldsymbol{y} \in C$ such that $\|\boldsymbol{x}-\boldsymbol{y}\|_2 \leq \frac{\varepsilon}{2} \leq \frac{1}{2}\left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$.*

Proof. Let $r_0 = \frac{1}{2}\left(\frac{2C_{\delta,n}\sqrt{d\log(n)}}{n\lambda}\right)^{1/d}$. Then we obtain $\int_{\mathcal{X}} f(\boldsymbol{z})\cdot\mathbb{1}_{[\|\boldsymbol{z}-\boldsymbol{x}\|_\infty \leq r_0]}d\boldsymbol{z} \geq (2r_0)^d(\lambda - R_2\cdot r_0^\beta) \geq (2r_0)^d\cdot\frac{\lambda}{2} = C_{\delta,n}\frac{\sqrt{d\log(n)}}{n}$, where the second inequality follows for sufficiently large n. By Lemma 2, there exists a point $\boldsymbol{y} \in X$ such that $\|\boldsymbol{x}-\boldsymbol{y}\|_\infty \leq r_0$. Comparing the order with respect to n (similar to the one in the proof of Lemma 3), we see that $r_0 \leq \frac{\varepsilon}{2\sqrt{d}}$. We therefore conclude that $\|\boldsymbol{x}-\boldsymbol{y}\|_2 \leq \frac{\varepsilon}{2} \leq \frac{1}{2}\left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$, implying from Lemma 4 that $\boldsymbol{y}$ is in C. ■

Now, we bound the Hausdorff error when using the core points returned by Algorithm 2 to estimate the desired density level set.

Definition 5 (Hausdorff Distance)**.** *Given two sets C and C', $\mathsf{dist}_{\mathsf{Haus}}(C, C') := \max\{\sup_{\boldsymbol{x}\in C}\mathsf{dist}(\boldsymbol{x}, C'), \sup_{\boldsymbol{x}'\in C'}\mathsf{dist}(\boldsymbol{x}', C)\}$.*

Theorem 3. *Let C be the set of core points obtained from Algorithm 2. With probability at least $1-\delta$, it follows that $\mathsf{dist}_{\mathsf{Haus}}(C, L_f(\lambda)) \leq 2\left(\frac{\lambda}{R_1}\cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$.*

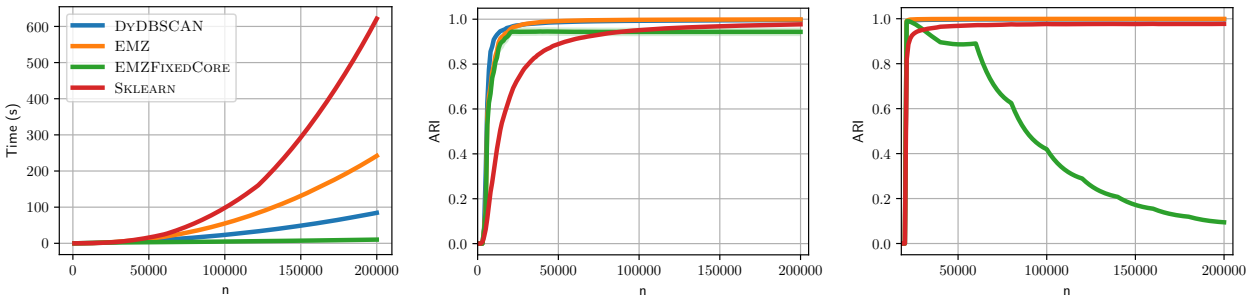
Proof. First, notice that Lemma 5 implies that $\sup_{\boldsymbol{x}\in L_f(\lambda)}\mathsf{dist}(\boldsymbol{x}, C) \leq \frac{1}{2}\left(\frac{\lambda}{R_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$. In addition, Lemma 3 implies that $\sup_{\boldsymbol{x}\in C}\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \leq 2\left(\frac{\lambda}{R_1}\cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$. Given the hyperparameter choice of k, $\mathsf{dist}_{\mathsf{Haus}}(C, L_f(\lambda)) \to 0$, as n tends to infinity. ■

Remark 3. *By selecting the maximum possible value for k, the resulting quantity is at most $\tilde{O}\left(M_3\cdot n^{-\frac{1}{2\beta+d}}\right)$, where M_3 is a parameter that only depends on (d, δ, λ), and the density function f. This matches with the lower bound established in Theorem 4 of Tsybakov (1997), indicating that our density level set estimation is near optimal.*

5 EMPIRICAL RESULTS

In this section, we evaluate the performance of our newly proposed dynamic DBSCAN algorithm by comparing it with several alternative methods. All experiments are executed on a single thread using a 13th Gen Intel(R) Core(TM) i5-13500 processor. We report the average results over 10 runs, with the standard error. The code to reproduce the experiments is available at `https://github.com/seiyun-shin/dynamic_dbscan`.

Experimental setup. We conduct evaluations on a variety of real-world and synthetic datasets, with their properties summarized in Table 1. The blobs dataset is a synthetic dataset drawn from a mixture of Gaussians. The other datasets are widely used for evaluating clustering and classification algorithms and are available on the OpenML dataset repository. Detailed descriptions



(a) (b) (c)

Figure 2: Comparison on the blobs dataset. (a) The running time of each algorithm. (b) The ARI for each algorithm when data points are added in a random order. (c) The ARI for each algorithm when data points are added cluster-by-cluster.

Dataset	Metric	Algorithm		
		DyDBSCAN	EMZ	Sklearn
Letter	Time	1.44±0.036	1.51±0.011	**1.10**±0.046
	ARI	**0.02**±0.001	**0.02**±0.002	0.00±0.000
	NMI	0.27±0.007	**0.29**±0.006	0.20±0.000
MNIST	Time	1.64±0.027	2.63±0.018	**0.99**±0.004
	ARI	**0.02**±0.001	**0.02**±0.002	0.00±0.000
	NMI	0.22±0.011	**0.26**±0.013	0.20±0.000
Fashion-MNIST	Time	**6.49**±0.159	20.55±0.057	26.17±0.051
	ARI	**0.05**±0.001	**0.05**±0.001	0.00±0.000
	NMI	0.15±0.003	**0.26**±0.001	0.05±0.000
Blobs	Time	**84.39**±1.008	241.96±2.943	621.43±1.921
	ARI	**1.00**±0.001	**1.00**±0.000	0.98±0.000
	NMI	0.99±0.001	**1.00**±0.000	0.97±0.000
KDDCup99	Time	**431.81**±3.975	6005.93±22.759	-
	ARI	**0.91**±0.001	**0.91**±0.000	-
	NMI	**0.80**±0.001	**0.80**±0.001	-
Covertype	Time	**874.01**±10.841	4073.99±22.875	-
	ARI	**0.05**±0.000	**0.05**±0.000	-
	NMI	**0.20**±0.000	**0.20**±0.000	-

Table 2: Experimental Results

and license information for these datasets can be found on the OpenML website (Vanschoren et al., 2013).

For the MNIST, Fashion-MNIST, and KDDCup99 datasets, we apply principal component analysis (PCA) to reduce the dimensionality to 20. For all datasets, we scale each dimension to have zero mean and unit variance. In every experiment, we dynamically stream the data points to the algorithm in a random order, with a batch size of 1000. After processing each batch, we compute the cluster labels for the entire dataset and evaluate the performance using two metrics: (1) the Adjusted Rand Index (ARI) and (2) Normalized Mutual Information (NMI).

Evaluation methods. To the best of our knowledge, our algorithm is the first dynamic version of DBSCAN, so we compare it with several simple baselines. Specifically, the algorithms we evaluate are as follows:

1. DynamicDBSCAN: the dynamic DBSCAN algorithm proposed in this paper.
2. EMZ: the near-linear time DBSCAN variant introduced by Esfandiari et al. (2021), where hash values for incoming points are computed once, and the graph is recomputed after processing each batch.
3. EMZFixedCore: a variant of the EMZ algorithm which we propose and describe later.
4. Sklearn: the standard DBSCAN implementation from the scikit-learn machine learning library (Pedregosa et al., 2011b).

For both DynamicDBSCAN and EMZ, we set the hyperparameters $k = 10$ and $t = 10$ and $\varepsilon = 0.75$. We observe that the hyperparameters k and t are not sensitive, noting that their theoretical values of $O(\log(n))$ changes slowly with the number of data points. Hence, setting k and t to their theoretical values would not significantly change the algorithm's performance.

Results and analysis. We report the results in Table 2. Due to the large memory requirement, we were unable to run the Sklearn algorithm on the KDDCup99 and Covertype datasets. From these results, it is evident that the DynamicDBSCAN algorithm achieves a faster running time on dynamic datasets, with a similar performance when compared with the baseline algorithms.

Comparison with a fixed core point set. In addition to the DynamicDBSCAN algorithm, we propose another variant, EMZFixedCore, which builds upon the EMZ algorithm. The EMZFixedCore algorithm processes the initial batch of data using the EMZ method, after which it keeps the set of core points fixed. For subsequent updates, the algorithm treats each arriving point as a non-core point and assigns it to the same cluster as the first core point it collides

with, determined by applying hash functions.

We conduct experiments on the blobs dataset under two scenarios. In the first scenario, data points were added in a random order. In the second, data points were added cluster-by-cluster, meaning that all points from cluster 1 were added before those from cluster 2, and so on. We found that the running time of the algorithms is unaffected by the order of the incoming data points. Figure 2 demonstrates that the EMZFixedCore algorithm exhibits a similar running time to DynamicDBSCAN and performs well when the order of the arriving data points is fully randomized. On the other hand, when each cluster is added one at a time, the EMZFixedCore algorithm struggles to handle the increasing number of clusters, resulting in significantly poorer performance compared to DynamicDBSCAN.

6 DISCUSSION

To the best of our knowledge, we propose the first dynamic version of the DBSCAN, capable of efficiently handling evolving datasets with poly-logarithmic time complexity with respect to the total number of updates.

For future work, one intriguing direction would be to explore whether further reductions in the logarithmic dependency can be achieved. Another promising direction is the development of a dynamic version of Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) (Campello et al., 2013).

Acknowledgments

The work of Ilan Shomorony was supported in part by the National Science Foundation (NSF) under grant CCF-2046991.

References

Baselice, F., Coppolino, L., D'Antonio, S., Ferraioli, G., and Sgaglione, L. (2015). A dbscan based approach for jointly segment and classify brain mr images. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2993–2996. IEEE.

Blackard, J. A. and Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151.

Campello, R. J., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 160–172. Springer.

Chaudhuri, K. and Dasgupta, S. (2010). Rates of convergence for the cluster tree. *Advances in neural information processing systems*, 23.

Chen, D. Z., Smid, M., and Xu, B. (2005). Geometric algorithms for density-based data clustering. *International Journal of Computational Geometry & Applications*, 15(03):239–260.

de Berg, M., Gunawan, A., and Roeloffzen, M. (2019). Faster dbscan and hdbscan in low-dimensional euclidean spaces. *International Journal of Computational Geometry & Applications*, 29(01):21–47.

Demaine, E. (2012). Advanced data structures. *Lecture Notes from MIT, Lecture 20*.

Esfandiari, H., Mirrokni, V., and Zhong, P. (2021). Almost linear time density level set estimation via dbscan. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7349–7357.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231.

Frey, P. W. and Slate, D. J. (1991). Letter recognition using holland-style adaptive classifiers. *Machine learning*, 6:161–182.

Gan, J. and Tao, Y. (2017). On the hardness and approximation of euclidean dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–45.

Gunawan, A. and de Berg, M. (2013). A faster algorithm for dbscan. *Master's thesis*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Henzinger, M. R. and King, V. (1995). Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 519–527.

Jang, J. and Jiang, H. (2019). Dbscan++: Towards fast and scalable density clustering. In *International conference on machine learning*, pages 3019–3029. PMLR.

Jiang, H. (2017). Density level set estimation on manifolds with dbscan. In *International Conference on Machine Learning*, pages 1684–1693. PMLR.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011a). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011b). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rinaldo, A. and Wasserman, L. (2010). Generalized density clustering. *The Annals of Statistics*, 38(5):2678–2722.

Schubert, E., Koos, A., Emrich, T., Züfle, A., Schmid, K. A., and Zimek, A. (2015). A framework for clustering uncertain data. *Proceedings of the VLDB Endowment*, 8(12):1976–1979.

Shen, J., Hao, X., Liang, Z., Liu, Y., Wang, W., and Shao, L. (2016). Real-time superpixel segmentation by dbscan clustering algorithm. *IEEE transactions on image processing*, 25(12):5933–5942.

Stolfo, S., Fan, W., Lee, W., Prodromidis, A., and Chan, P. (1999). Kdd cup 1999 data. A dataset used for computer network intrusion detection, containing simulated intrusions in a military network environment.

Team, R. C. et al. (2013). R: A language and environment for statistical computing. *Foundation for Statistical Computing, Vienna, Austria*.

Tran, T. N., Nguyen, T. T., Willemsz, T. A., van Kessel, G., Frijlink, H. W., and van der Voort Maarschalk, K. (2012). A density-based segmentation for 3d images, an application for x-ray micro-tomography. *Analytica chimica acta*, 725:14–21.

Tseng, T., Dhulipala, L., and Blelloch, G. (2019). Batch-parallel euler tour trees. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 92–106. SIAM.

Tsybakov, A. B. (1997). On nonparametric estimation of density level sets. *The Annals of Statistics*, 25(3):948–969.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2013). Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A Proof of Lemma 1

Although the proof of Lemma 1 is in Esfandiari et al. (2021), we provide the full proof for completeness:

Lemma 6 ((Esfandiari et al., 2021)). *Given $\varepsilon > 0$, the following holds for any two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ and for a hash function h:*

1. $\Pr[h(\boldsymbol{x}) = h(\boldsymbol{y})] \geq 1 - \frac{\|\boldsymbol{x}-\boldsymbol{y}\|_1}{2\varepsilon}$;
2. $h(\boldsymbol{x}) = h(\boldsymbol{y}) \implies \|\boldsymbol{x} - \boldsymbol{y}\|_\infty \leq 2\varepsilon$.

Proof. Fix a coordinate $j \in [d]$. Observe that $\Pr[\lfloor \frac{\boldsymbol{x}_j+\eta}{2\varepsilon} \rfloor \neq \lfloor \frac{y_j+\eta}{2\varepsilon} \rfloor]$ is at most $\frac{|\boldsymbol{x}_j - y_j|}{2\varepsilon}$. By applying a union bound over all coordinates, we obtain $\Pr[h(\boldsymbol{x}) \neq h(\boldsymbol{y})] \leq \frac{\|\boldsymbol{x}-\boldsymbol{y}\|_1}{2\varepsilon}$, proving the first part.

Now, suppose $\|\boldsymbol{x} - \boldsymbol{y}\|_\infty > 2\varepsilon$. Then there must exist a coordinate $j \in [d]$ such that $|\boldsymbol{x}_j - y_j| > 2\varepsilon$. This implies that $\lfloor \frac{\boldsymbol{x}_j+\eta}{2\varepsilon} \rfloor \neq \lfloor \frac{y_j+\eta}{2\varepsilon} \rfloor$ for any $\eta > 0$, which contradicts the assumption that $h(\boldsymbol{x}) = h(\boldsymbol{y})$. Hence, we conclude that if $h(\boldsymbol{x}) = h(\boldsymbol{y})$, then $\|\boldsymbol{x} - \boldsymbol{y}\|_\infty \leq 2\varepsilon$, proving the second part of the lemma. This completes the proof of Lemma 1. ■

B Deferred Details in 3 via Order-wise Comparison

In this section we present detailed derivations of the claim referenced in the proof of Lemma 3. Specifically, we aim to prove the following:

Claim 1. *For any point $\boldsymbol{x} \in X$ with $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \geq 2\left(\frac{\lambda}{R_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{\frac{1}{\beta}}$, it holds for sufficiently large n that $\mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) \geq 4\sqrt{d}\varepsilon$.*

This claim is critical to the proof of Lemma 3, as it shows that, given $\mathsf{dist}(\boldsymbol{y}, L_f(\lambda)) \geq \mathsf{dist}(\boldsymbol{x}, L_f(\lambda)) - 2\sqrt{d}\varepsilon$, we can conclude $\mathsf{dist}(\boldsymbol{y}, L_f(\lambda)) \geq \frac{1}{2} \cdot \mathsf{dist}(\boldsymbol{x}, L_f(\lambda))$. To prove the claim, we will consider two cases: (1) $d = \Theta(1)$ and (2) $d = \Theta(\log^c(n))$ for any constant $c > 0$.

B.1 Proof of Claim 1 for $d = \Theta(1)$

Taking d to be a constant, we will show that

$$\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} = \left(\frac{1}{M_2}\right)^{\frac{1}{2\beta}} \cdot \omega\left((\log(n)/n)^{\frac{1}{2\beta+d}}\right) \text{ and } 2\sqrt{d}\varepsilon = M_2^{\frac{1}{d}} \cdot O\left((\log(n)/n)^{\frac{1}{2\beta+d}}\right).$$

Notice the parameter setup mentioned in Section 4.3.1 are:

- $k \geq M_1 \cdot \left(\frac{\lambda}{\lambda_c}\right)^2 \cdot \omega(\log(n) \cdot \log^{2+\alpha}\log(n))$ for some $\alpha > 0$;
- $k \leq M_2 \cdot \lambda^{\frac{2\beta+2d}{2\beta+d}} \cdot O\left((\log(n))^{\frac{d}{2\beta+d}} \cdot n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma}\log(n)\right)$ for some $\gamma > 0$;
- $\varepsilon := \frac{1}{2}\left(\frac{k}{n\lambda(1-2C_{\delta,n}/\sqrt{k})}\right)^{\frac{1}{d}}$;
- $t := 100\log\left(\frac{n}{\delta}\right)$;

where $C_{\delta,n} := C_0\left(\log(t/\delta)\sqrt{d\log(n)}\right)$.

First notice that we have:

$$\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} \geq \left(\frac{\lambda}{R_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{\frac{1}{\beta}}.$$

Substituting the upper bound for $k \leq M_2 \cdot \lambda^{\frac{2\beta+2d}{2\beta+d}}(\log(n))^{\frac{d}{2\beta+d}} \cdot n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma}\log(n)$, we obtain:

$$\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} \geq \left(\frac{10C_0\sqrt{d\log(n)}\log(t/\delta) \cdot \log^{\frac{\gamma}{2}}\log(n)}{R_1\sqrt{M_2}\lambda^{\frac{\beta+d}{2\beta+d}}(\log(n))^{\frac{d}{2(2\beta+d)}}n^{\frac{\beta}{2\beta+d}}}\right)^{\frac{1}{\beta}}.$$

Notice that the exponent of $\log(n)$ simplifies to:

$$\left(\frac{1}{2} - \frac{d}{2(2\beta+d)}\right) \cdot \frac{1}{\beta} = \left(\frac{2\beta+d-d}{2(2\beta+d)}\right) \cdot \frac{1}{\beta} = \frac{1}{2\beta+d},$$

which will lead us to obtain the order of $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$ as:

$$\left(\frac{1}{M_2}\right)^{\frac{1}{2\beta}} \cdot \omega\left(\left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}} \cdot \log^{\frac{\gamma}{2\beta}} \log(n)\right).$$

For $2\sqrt{d}\varepsilon$, we have:

$$2\sqrt{d}\varepsilon = \sqrt{d}\left(\frac{k}{n\lambda(1 - 2C_{\delta,n}/\sqrt{k})}\right)^{\frac{1}{d}}.$$

Substituting the upper bound for k, we obtain:

$$2\sqrt{d}\varepsilon \leq \sqrt{d}\left(\frac{M_2 \lambda^{\frac{2\beta+2d}{2\beta+d}} (\log(n))^{\frac{d}{2\beta+d}} n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma} \log(n)}{n\lambda(1 - 2C_{\delta,n}/\sqrt{k})}\right)^{\frac{1}{d}}.$$

Observe that using the lower bound for k, we can estimate:

$$\frac{C_{\delta,n}}{\sqrt{k}} \leq \frac{C_0\left(\log(t/\delta)\sqrt{d\log(n)}\right)}{\sqrt{\log(n) \cdot \log^{2+\alpha} \log(n)}} = O\left(\frac{1}{\log^{\frac{\alpha}{2}} \log(n)}\right) \to 0, \quad \text{as} \quad n \to \infty.$$

Hence approximating for small $\frac{C_{\delta,n}}{\sqrt{k}}$ for sufficiently large n, we simplify:

$$2\sqrt{d}\varepsilon \leq \sqrt{d}\left(\frac{M_2 \lambda^{\frac{2\beta+2d}{2\beta+d}-1} (\log(n))^{\frac{d}{2\beta+d}} n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma} \log(n)}{n}\right)^{\frac{1}{d}}.$$

As the exponent of n becomes $\left(\frac{2\beta}{2\beta+d} - 1\right) \cdot \frac{1}{d} = -\frac{1}{2\beta+d}$, the order of $2\sqrt{d}\varepsilon$ is then:

$$M_2^{\frac{1}{d}} \cdot O\left(\left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}} \cdot \log^{-\frac{\gamma}{d}} \log(n)\right).$$

In conclusion, both $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$ and $2\sqrt{d}\varepsilon$ have the same asymptotic behaviour with respect to n, which is $\Theta\left(\left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}}\right)$; however, the constant factor for $2\sqrt{d}\varepsilon$ is dependent on $M_2^{\frac{1}{d}}$, while for $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$, it involves $\left(\frac{1}{M_2}\right)^{\frac{1}{2\beta}}$. Consequently, if we choose M_2 sufficiently small and n large enough, we can ensure $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} > 2\sqrt{d}\varepsilon$. This completes the proof. ■

B.2 Proof of Claim 1 for $d = \Theta(\log^c(n))$ and Constant $c > 0$

In this subsection, we analyse the order-wise comparison between $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$ and $2\sqrt{d}\varepsilon$ with respect to d and n. To account for the dependence on d, we use the following detailed parameter setup:

- $100 \cdot (100\sqrt{d})^{2\beta+d} \left(\frac{\lambda}{\lambda_c}\right)^2 \left(\frac{R_2}{R_1}\right)^2 \cdot C_{\delta,n}^2 \sqrt{d\log(n)} \cdot \log^{2+\alpha} \log(n) \leq k \leq \left(\frac{C_{\delta,n}}{R_2}\right)^{\frac{2d}{2\beta+d}} \left(\frac{1}{4d}\right)^{\frac{\beta d}{2\beta+d}} \lambda^{\frac{2\beta+2d}{2\beta+d}} n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma} \log(n)$ for some $\alpha, \gamma > 0$;
- $\varepsilon := \frac{1}{2}\left(\frac{k}{n\lambda(1-2C_{\delta,n}/\sqrt{k})}\right)^{\frac{1}{d}}$; and

- $t := 100 \log\left(\frac{n}{\delta}\right)$;

where $C_{\delta,n} = C_0\left(\log(t/\delta)\sqrt{d\log(n)}\right)$. Using the upper bound for k, we now substitute into the expression for $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$, yielding:

$$\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} \geq \left(\frac{\lambda}{R_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{\frac{1}{\beta}} \geq \left(\frac{10}{R_1} \cdot \frac{C_0 \log(t/\delta)\sqrt{d\log(n)} \cdot \log^{\frac{\gamma}{2}} \log(n)}{\left(\frac{C_0 \log(t/\delta)(d\log(n))^{1/2}}{R_2}\right)^{\frac{d}{2\beta+d}} \cdot \left(\frac{1}{4d}\right)^{\frac{\beta d}{2(2\beta+d)}} \lambda^{\frac{\beta+d}{2\beta+d}} n^{\frac{\beta}{2\beta+d}}}\right)^{\frac{1}{\beta}}.$$

After simplifying the terms involving d, $\log\log(n)$, $\log(n)$, and n, the expression becomes:

$$\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} = \omega\left(d^{\frac{2+d}{2(2\beta+d)}} \cdot \left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}} \cdot \log^{\frac{\gamma}{2\beta}} \log(n)\right),$$

where the exponent for $\frac{\log(n)}{n}$ and $\log\log(n)$ follow from the same analysis as for $d = \Theta(1)$, and the the exponent for d comes from:

$$\left(\frac{1}{2} - \frac{d}{2(2\beta+d)} + \frac{\beta d}{2(2\beta+d)}\right) \cdot \frac{1}{\beta} = \left(\frac{2\beta + d - d + \beta d}{2(2\beta+d)}\right) \cdot \frac{1}{\beta} = \frac{2+d}{2(2\beta+d)}.$$

Next, using the approximation $1 - 2C_{\delta,n}/\sqrt{k} \approx 1$ for sufficiently large n, we simplify: $2\sqrt{d}\varepsilon \approx \sqrt{d}\left(\frac{k}{n\lambda}\right)^{\frac{1}{d}}$. Substituting the upper bound for k into the expression for ε, we obtain:

$$2\sqrt{d}\varepsilon = \sqrt{d}\left(\left(\frac{1}{n\lambda}\right) \cdot \left(\frac{C_0 \log(1/\delta)\sqrt{d\log(n)}}{R_2}\right)^{\frac{2d}{2\beta+d}} \cdot \left(\frac{1}{4d}\right)^{\frac{\beta d}{2\beta+d}} \cdot \lambda^{\frac{2\beta+2d}{2\beta+d}} \cdot n^{\frac{2\beta}{2\beta+d}} \cdot \log^{-\gamma} \log(n)\right)^{\frac{1}{d}}.$$

Simplifying further, we have:

$$2\sqrt{d}\varepsilon = O\left(d^{\frac{2+d}{2(2\beta+d)}} \cdot \left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}} \cdot \log^{-\frac{\gamma}{d}} \log(n)\right),$$

where the exponent of d arises from:

$$\frac{1}{2} + \left(\frac{1}{2} \cdot \frac{2d}{2\beta+d} - \frac{\beta d}{2\beta+d}\right) \cdot \frac{1}{d} = \frac{1}{2} + \frac{1-\beta}{2\beta+d} = \frac{2\beta + d + 2(1-\beta)}{2(2\beta+d)} = \frac{2+d}{2(2\beta+d)}.$$

Therefore, both $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2}$ and $2\sqrt{d}\varepsilon$ exhibit the same asymptotic behaviour with respect to d and n, namely:

$$d^{\frac{2+d}{2(2\beta+d)}} \cdot \left(\frac{\log(n)}{n}\right)^{\frac{1}{2\beta+d}},$$

except for the $\log\log n$ factor. The presence of the $\log\log n$ dependency implies that for sufficiently large n, we can ensure $\frac{\mathsf{dist}(\boldsymbol{x}, L_f(\lambda))}{2} > 2\sqrt{d}\varepsilon$.

Lastly, we highlight that the regime of $d = \Theta(\log^c(n))$ for a constant $c > 0$ is essential for ensuring that the volume of the cubes with side-length proportional to ε used in density level estimation remains bounded, even when n grows large. This completes the proof. ■