Approximate DBSCAN via Density-Biased Sampling and Kernel Density Estimation

JIAN LIN, College of Computer Science and Software Engineering, Shenzhen University, China
SIYUE WU, College of Computer Science and Software Engineering, Shenzhen University, China
DINGMING WU*, College of Computer Science and Software Engineering, Shenzhen University, China
TSZ NAM CHAN, College of Computer Science and Software Engineering, Shenzhen University, China

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular unsupervised algorithm that identifies clusters as dense regions separated by sparse areas. It requires no preset number of clusters, can detect arbitrary shapes, and is robust to noise, making it widely adopted. However, exact DBSCAN suffers from efficiency issues, and existing approximate variants often trade accuracy for speed. To address this, we reformulate DBSCAN as a Minimum Connected Dominating Set (MCDS) problem and propose LDBS-KDE, a fast and accurate approximation algorithm. LDBS-KDE selects candidate points via lattice-based density-biased sampling, identifies core points using kernel density estimation, and constructs clusters by grouping cores and assigning remaining points. Extensive experiments on four real-world datasets demonstrate that LDBS-KDE achieves accuracy comparable to, and in some cases surpassing, state-of-the-art methods, while offering an improvement in efficiency by a factor of two up to two orders of magnitude.

CCS Concepts: • **Theory of computation → Approximation algorithms analysis**; • **Information systems** → *Clustering*.

Additional Key Words and Phrases: Approximate DBSCAN; Minimum Connected Dominating Set; Kernel Density Estimation; Sampling

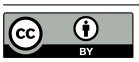
ACM Reference Format:
Jian Lin, Siyue Wu, Dingming Wu, and Tsz Nam Chan. 2026. Approximate DBSCAN via Density-Biased Sampling and Kernel Density Estimation. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 136 (June 2026), 24 pages. https://doi.org/10.1145/3802013

1 Introduction

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [16] is a widely used unsupervised clustering algorithm that identifies clusters as dense regions separated by sparse areas, labeling points in low-density regions as outliers. It is parameterized by the neighborhood radius ϵ and the minimum number of points *MinPts* required to form a dense region, with clusters constructed by connecting core points that satisfy these conditions. DBSCAN is advantageous in that it does not require the number of clusters to be specified in advance, can discover clusters of arbitrary shape, and is robust to noise. These properties have led to its broad adoption in image

*Corresponding author

Authors' Contact Information: Jian Lin, linjian.bit@gmail.com, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China; Siyue Wu, wusiyue1229@gmail.com, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China; Dingming Wu, dingming.wu@szu.edu.cn, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China; Tsz Nam Chan, edisonchan@szu.edu.cn, College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China.



ACM 2836-6573/2026/6-ART136
https://doi.org/10.1145/3802013

processing [15], geospatial analysis [24], autonomous robot navigation [29], and behavioral pattern mining [44].

However, it has been proven that the lower bound on the time complexity of exact DBSCAN algorithms is $\Omega(n^{4/3})$ [18, 19] for data of dimensionality higher than two, which limits their applicability to large-scale datasets. To alleviate this issue, various approximate DBSCAN algorithms [7, 10–12, 14, 18, 19, 25, 26, 30, 39, 43] have been proposed; however, existing approximate algorithms often suffer from either inefficiency or limited accuracy.

Efficiency limitation. Table 1 summarizes the time complexities of existing approximate algorithms. Given a d-dimensional dataset with n points, LSH-DBSCAN [39] has quadratic complexity, while RNN-DBSCAN [7] runs in superlinear time. ρ-DBSCAN [18, 19] and GRIDEN [14] both have a time complexity of $O(n \cdot c^d)$, exponential in the data dimensionality. KNN-DBSCAN [10], BLOCK-DBSCAN [11], KNN-BLOCK DBSCAN [12], NaGB-DBSCAN [30], DBSCAN++ [25], and SNG-DBSCAN [26] run in $O(n \log n)$ time. Although the most efficient variant to date, sDBSCAN [43], achieves linear complexity in both n and d, its accuracy is limited. Specifically, to preserve the fidelity of the original DBSCAN clustering structure with high probability $(1 - \delta)$, the number of random projections required by sDBSCAN grows linearly with the dataset size and exponentially with the inverse error probability $\log(1/\delta)$. This requirement becomes particularly demanding when the clustering structure exhibits low density.

Table 1. Time complexities of approximate DBSCAN.

Algorithm	Time Complexity
LSH-DBSCAN [39]	$O(n^2)$
RNN-DBSCAN [7]	$O(n^{1.3})$
ρ-DBSCAN [18, 19]	$O(n \cdot c_1^{d-1})$, $c_1 = 1 + 1/\rho$
GRIDEN [14]	$O(n \cdot c_2^d)$, $c_2 = 2g + 1$
KNN-DBSCAN [10]	$O(n \log n + n \cdot M)$
BLOCK-DBSCAN [11]	$O(n \log n)$
KNN-BLOCK DBSCAN [12]	$O(Ld \cdot n \log n / \log \mathcal{X})$
NaGB-DBSCAN [30]	$O(n \log n)$
DBSCAN++ [25]	$O(n \log n + n \cdot s)$
SNG-DBSCAN [26]	$O(n \log n)$
sDBSCAN [43]	$O(n \cdot k + n \cdot dD \log D)$
LDBS-KDE (ours)	$O\big(n(d+s) + \lambda \log \lambda + s \log s + s\lambda\big)$

$\star$ ρ the approximation error, g the grid cell division degree, M the maximum number of points in a data point's neighborhood, k the number of random vectors per data point, D the total number of random vectors, L the number of examined points specified by the user, $\mathcal{X}$ the branching factor, s the number of samples, and λ the sample size of the approximate KDE algorithm.

Accuracy limitation. Existing approximate algorithms either provide no theoretical guarantees on clustering quality or rely on strong assumptions to ensure accuracy. LSH-DBSCAN [39], KNN-BLOCK DBSCAN [12], and NaGB-DBSCAN [30] do not provide theoretical guarantees on their approximation error, so the quality of the resulting clusters cannot be assured. Similarly, BLOCK-DBSCAN [11] and KNN-DBSCAN [10] recover only subsets of the true clusters, without any guarantee on the overall clustering accuracy. ρ-DBSCAN [18, 19] and GRIDEN [14] return clusters that lie between those produced by the exact DBSCAN with parameter settings $(\epsilon, MinPts)$ and $(\delta \cdot \epsilon, MinPts)$, where $\delta > 1$. Consequently, nearby true clusters may be incorrectly merged into a single cluster. DBSCAN++ [25] and SNG-DBSCAN [26] quantify the approximation error using

the Hausdorff distance [22] between the estimated and true clustering results. However, their analyses rely on strong assumptions about the underlying data distribution, which limits their applicability in practical scenarios. sDBSCAN [43] can recover the true clusters only when the common neighborhood of any two nearby core points contains at least $O(\log n_i)$ core points, where n_i denotes the number of core points in the true cluster C_i. However, this condition may not always be satisfied in real-world datasets.

In this work, we propose a novel efficient approximate method for generating accurate DBSCAN clusters on large-scale datasets. Existing approaches typically focus on approximating ϵ-neighborhood queries [7, 10, 39, 43], detecting dense regions at the grid-cell or block level [11, 12, 14, 18, 19, 30, 31], or computing clusters from sampled points [25, 26] using uniform or greedy k-center sampling [20]. In contrast, we reformulate DBSCAN clustering as a Minimum Connected Dominating Set (MCDS) problem on the Cluster Graph (CGraph). This perspective is crucial because DBSCAN cluster connectivity does not require all core points: a connected dominating subset of core points is sufficient to preserve both density reachability and cluster structure. Consequently, the problem of DBSCAN clustering can be reduced to identifying a small representative subset of core points whose induced connectivity recovers the original clusters. However, computing an exact MCDS is NP-hard and the CGraph is unknown prior to clustering, making solutions of MCDS impractical for large-scale datasets. This observation motivates LDBS-KDE, a fast and accurate approximation algorithm that explicitly targets the MCDS structure rather than ϵ-neighborhood enumeration. The approximation error enters at two controlled stages. First, instead of searching over all points, LDBS-KDE applies lattice-based density-biased sampling to select candidate points. This step approximates the dominating set by preferentially sampling dense regions and structural connectors, thereby preserving cluster topology while drastically reducing the search space. Second, core points are identified from the sampled candidates using kernel density estimation (KDE), which approximates local density without explicit range queries, trading exact neighborhood counts for efficient density estimates. Once approximate core points are obtained, LDBS-KDE constructs intermediate clusters by grouping connected core points, which captures the global DBSCAN cluster structure from a small sample. Finally, all remaining points are assigned to clusters via nearest-core association, completing cluster recovery. By avoiding exhaustive neighborhood computation and focusing on a compact structural backbone, LDBS-KDE runs in time linear in the dataset size n and the sample size s ($s \ll n$). Extensive evaluations on four real-world datasets show that the proposed algorithm, LDBS-KDE, achieves accuracy comparable to, and in some cases surpassing, state-of-the-art methods, while offering an improvement in efficiency by a factor of two up to two orders of magnitude.

Contributions. Our contributions are summarized as follows:

- We reformulate DBSCAN clustering as a Minimum Connected Dominating Set (MCDS) problem, showing that cluster recovery can be achieved using a substantially reduced subset of core points.
- To address the NP-hardness of MCDS, we propose a fast and accurate approximation algorithm LDBS-KDE. The algorithm employs lattice-guided density-biased sampling to select candidate points and leverages kernel density estimation (KDE) to efficiently identify core points, avoiding expensive range queries. By constructing clusters from sampled core points and assigning remaining points via nearest-core association, LDBS-KDE preserves intrinsic clustering structure while achieving linear time complexity in the dataset size and sample size.
- Empirical results on four real-world datasets indicate that LDBS-KDE delivers comparable or superior accuracy to existing methods, with efficiency gains from 2× up to two orders of magnitude.

The rest of the paper is organized as follows: Section 2 reviews related algorithms, while Section 3 introduces the concepts of DBSCAN and kernel density estimation. Section 4 reformulates the DBSCAN clustering problem and presents the proposed approximate algorithm, LDBS-KDE. Extensive experimental evaluations are reported in Section 5, and conclusions are drawn in Section 6.

2 Related Work

Approximate DBSCAN Algorithms. To improve efficiency, several approximate variants of DBSCAN have been proposed, which can be categorized into four groups.

Grid-based DBSCAN. ρ-DBSCAN [18, 19] uses a quadtree-like grid index for approximate range queries, and GRIDEN [14] clusters at the cell level with a tunable granularity parameter. All grid-based methods suffer exponential complexity growth with dimension d.

Neighborhood-based DBSCAN. These methods accelerate clustering by approximating ϵ-neighborhoods. LSH-DBSCAN [39] detects core points via locality-sensitive hashing, RNN-DBSCAN [7] uses reverse nearest neighbors, and KNN-DBSCAN [10] leverages approximate k-NN graphs. While effective, neighborhood computation remains the main bottleneck, with complexities ranging from superlinear to quadratic in high dimensions. sDBSCAN [43] identifies core points with random projections. Although it achieves linear time, $O(n \cdot k + n \cdot dD \log D)$, maintaining acceptable accuracy requires large values of D and k.

Block-based DBSCAN. These algorithms cluster at the block level to reduce redundant density computations. BLOCK-DBSCAN [11], KNN-BLOCK DBSCAN [12], and NaGB-DBSCAN [30] use various block or granule structures with approximate neighborhood searches. However, block sizes are tied to ϵ, limiting computational savings; overall complexity remains roughly $O(n \log n)$.

Sampling-based DBSCAN. Using uniform or greedy k-center sampling [20], DBSCAN++ [25] and SNG-DBSCAN [26] compute neighborhoods for sampled points to reduce cost. While they improve efficiency, oversampling dense regions and undersampling sparse ones can degrade clustering quality.

Our approach goes beyond existing approximate DBSCAN methods by reformulating DBSCAN itself rather than approximating ϵ-neighborhood queries. While prior grid-based, neighborhood-based, block-based, and sampling-based methods all focus on accelerating or approximating neighborhood computations—the main bottleneck in DBSCAN—we model clustering as a MCDS problem, observing that only a small, representative subset of core points is sufficient to recover cluster connectivity. This shifts the approximation target from neighborhoods to the structural backbone of clusters. To efficiently approximate this subset, we introduce lattice-based density-biased sampling that preserves cluster structure more effectively than uniform or k-center sampling, and we identify core points via kernel density estimation instead of range queries. By constructing clusters from a compact set of core points and assigning remaining points by nearest-core association, our method achieves linear scalability while maintaining accuracy comparable to or better than state-of-the-art approaches.

Approximate KDE Algorithms. Computing the Kernel Density Estimate (KDE) at a query point requires $O(n)$ time, which is costly for large datasets. To reduce this overhead, approximate KDE algorithms have been developed. Sampling-based methods [35–37, 42, 45] construct a coreset that yields an ε-approximation of the KDE with probability at least $1 - \delta$. DEANN [28] improves this by exactly computing contributions of high-impact points via ANN [27] and sampling the rest. While DEANN and Zrandom+RS [45] support a broad range of kernels, others [35–37, 42] focus on the Gaussian kernel and lack uniform kernel support. Hashing-based approaches aim to improve efficiency. HBE [9] uses LSH to estimate KDE values, and HBS [41] introduces non-uniform sampling to reduce preprocessing. Backurs et al. [5] cut preprocessing and space costs to linear via independent subsampling, while Charikar et al. [8] leverage data-dependent LSH to speed up

queries. However, these methods target rapidly decaying radial kernels and perform poorly with the uniform kernel. Random Fourier Features (RFF) [38] approximate kernels using a small set of random projections, and DMKDE [17] extends this with quantum-inspired density matrices for efficiency. These techniques work well for smooth radial kernels such as Gaussian and Laplace but also fail with the uniform kernel.

Our algorithm integrates any approximate KDE method that supports the uniform kernel and identifies core points in linear time. For experiments, we adopt Zrandom+RS [45], the state-of-the-art method in this setting.

3 Preliminaries

3.1 DBSCAN

Given a set of n data points P in a d-dimensional space $\mathbb{R}^d$, DBSCAN [16] clusters data points that are densely distributed and labels those in sparse regions as noise. DBSCAN has two parameters: (i) Parameter $\epsilon > 0$ defines the ϵ-neighborhood of a data point $p \in P$, denoted as $N_\epsilon(p) = \{q \in P \mid \|p - q\| \leq \epsilon\}$, where $\|p - q\|$ represents the Euclidean distance between p and q. (ii) Parameter $MinPts \in \mathbb{N}$ specifies the minimum number of data points required within an ϵ-neighborhood to form a dense region.

DBSCAN categorizes data points as core points, border points, or noise:

- A **core point** p has at least $MinPts$ neighbors within its ϵ-neighborhood, i.e., $|N_\epsilon(p)| \geq MinPts$.
- A **border point** has fewer than $MinPts$ neighbors but lies within the ϵ-neighborhood of a core point.
- A **noise point** is neither a core point nor a border point.

A data point q is **directly density-reachable** from a data point p with respect to parameters ϵ and $MinPts$ if $q \in N_\epsilon(p)$ and p is a core point. A data point q is **density-reachable** from p if there exists a sequence of points $p_1, p_2, \ldots, p_k$ such that $p_1 = p$, $p_k = q$, and p_{i+1} is directly density-reachable from p_i for $1 \leq i < k$. Two points p and q are **density-connected** if there exists a core point o such that both p and q are density-reachable from o. Density-connectivity is a symmetric relation that enables grouping points into the same cluster whenever they share a common core point from which both are density-reachable.

A DBSCAN **cluster** C w.r.t. ϵ and $MinPts$ is a non-empty subset of P that satisfies:

- **Maximality:** $\forall p, q$: if $p \in C$ and q is density-reachable from p, then $q \in C$.
- **Connectivity:** $\forall p, q \in C$, p is density-connected to q.

Problem statement. The approximate DBSCAN clustering problem aims to compute a clustering $C' = \{C'_1, C'_2, \ldots\}$ that approximates the exact DBSCAN clustering C under the same parameters $(\epsilon, MinPts)$, while reducing computational cost.

3.2 Kernel Density Estimation

Kernel Density Estimation (KDE) [33] is a non-parametric method for estimating a probability density function. It employs a kernel function $K_*(\cdot)$ to fit the observed data points, thereby approximating the underlying distribution. The kernel is scaled by a bandwidth parameter h, which controls the smoothness of the estimate: a small h produces a highly detailed but potentially overfitted density, whereas a large h yields a smoother approximation. Given a set P of n data points in a d-dimensional space $\mathbb{R}^d$, the KDE of the density at a query point $x \in \mathbb{R}^d$ using kernel function $K_*(\cdot)$ is defined as [40]:

$$\mathsf{KDE}_{P,*}(x) = \frac{1}{n \cdot h^d} \sum_{p \in P} K_* \left(\frac{\|x - p\|}{h} \right). \tag{1}$$

A commonly used kernel function is the uniform kernel $K_U(\cdot)$, which assigns equal weight to all data points within a fixed radius h of the query point and zero weight to all points outside this radius.

$$K_U(u) = \kappa_d \cdot \mathbb{I}(|u| \leq 1), \tag{2}$$

where κ_d represents the normalization constant of the kernel function in d-dimensional space.

Computing $\mathsf{KDE}_P(x)$ requires $O(n)$ time, which becomes inefficient when the dataset P is large. The approximate KDE algorithm [45] addresses this issue by returning an estimate $\widehat{\mathsf{KDE}}_Q(x)$ for each query point x based on a small point set Q (often $Q \subset P$, but not necessarily). Given an error bound B and parameter $\delta \in (0, 1)$, the approximate KDE algorithm produces Q such that:

$$\Pr\left[\max_{x \in \mathbb{R}^d} \left|\mathsf{KDE}_P(x) - \widehat{\mathsf{KDE}}_Q(x)\right| \leq B\right] \geq 1 - \delta. \tag{3}$$

4 Approximate Method for DBSCAN Clustering

To achieve fast and accurate DBSCAN approximation, we reformulate the DBSCAN clustering problem as a Minimum Connected Dominating Set (MCDS) problem (Section 4.1), enabling cluster recovery with a reduced number of core points. Since the MCDS problem is NP-hard, we propose the LDBS-KDE algorithm (Section 4.2) to compute an approximate solution efficiently. Section 4.3 analyzes the time complexity of the LDBS-KDE algorithm and Section 4.4 presents its approximation error analysis.

4.1 MCDS Formulation for DBSCAN Clustering

We first revisit the representation of DBSCAN clusters through the cluster graph (Definition 4.1).

Given a set of data points $P \subset \mathbb{R}^d$ and DBSCAN parameters ϵ and *MinPts*, let C be the set of DBSCAN clusters detected from P and $P^{\mathsf{core}} \subseteq P$ denote the set of core points in these clusters.

Definition 4.1 (Cluster Graph [43]). *The* Cluster Graph (CGraph) *of C is the undirected graph $G^C = (V^C, E^C)$, where $V^C \subseteq P$ is the set of all data points belonging to DBSCAN clusters, and:*

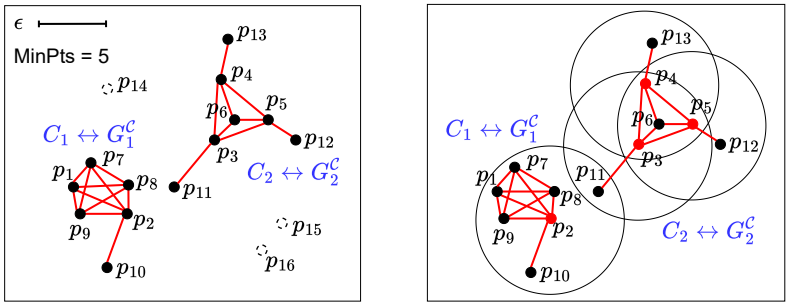
Case I: *an edge connects every pair of core points $p_i, p_j \in P^{\mathsf{core}}$ such that $\|p_i - p_j\| \leq \epsilon$;*

Case II: *for each non-core point $p \in V^C \setminus P^{\mathsf{core}}$, an edge connects p and one core point in its ϵ-neighborhood $N_\epsilon(p)$. Vertex p is called a leaf vertex.*

If C contains only one cluster, the CGraph G^C is connected. If C contains β clusters $C_1, C_2, \ldots C_\beta$, the CGraph G^C consists of β connected components, $G^C = \{G_i^C \mid 1 \leq i \leq \beta\}$. For each DBSCAN cluster $C_i \in C$, the vertex set of the corresponding connected component G_i^C is exactly the set of data points in C_i. Thus, every data point in a cluster has a vertex representation in a connected component, and vice versa.

Example 4.1. Figure 1a illustrates a set of 16 data points in $\mathbb{R}^2$. Using the DBSCAN parameters ϵ and *MinPts* indicated in the figure, two DBSCAN clusters are detected: $C_1 = \{p_1, p_2, p_7, p_8, p_9, p_{10}\}$ and $C_2 = \{p_3, p_4, p_5, p_6, p_{11}, p_{12}, p_{13}\}$, while points p_{14}, p_{15}, and p_{16} are classified as noise. The set of core points is $P^{\mathsf{core}} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$, and the remaining black vertices represent non-core points. In the same figure, the corresponding CGraph is also depicted, consisting of the black vertices and red edges. It contains two connected components, G_1^C and G_2^C, corresponding to clusters C_1 and C_2, respectively.

Next, we apply the minimum connected dominating set (MCDS) problem (Definition 4.2) to the CGraph and demonstrate that the DBSCAN clustering problem can be resolved by the solution to the MCDS problem (Lemma 4.2).



(a) CGraph (b) MCDS of CGraph

Fig. 1. Examples of the CGraph and its MCDS.

Definition 4.2 (Decision Version of the Minimum Connected Dominating Set (MCDS) Problem [21]). *Given an undirected connected graph $G = (V, E)$ and an integer k, does there exist a subset $D \subseteq V$ such that:*

(i) D is a dominating set of G, i.e., for every vertex $v \in V \setminus D$, there exists a vertex $u \in D$ such that $(u, v) \in E$.
(ii) the subgraph of G induced by D is connected;
(iii) $|D| \leq k$.

Given a CGraph G^C consisting of β connected components, let D_i denote the MCDS of the component G_i^C.

Lemma 4.1. *$\forall p \in D_i$, p is a core point.*

Proof. Assume, for contradiction, that D_i contains a non-core point p. By Case II of Definition 4.1, p is a leaf vertex and has exactly one core neighbor c within its ϵ-neighborhood. For D_i to induce a connected component G_i^C, c must also belong to D_i; otherwise, p would be isolated. Define $D_i' = D_i \setminus \{p\}$. Then, we have

(i) Any vertex $q \neq p$ dominated by D_i remains dominated by D_i', and p itself is dominated by $c \in D_i'$.
(ii) Since the component G_i^C induced by D_i is connected, removing the leaf vertex p leaves the component induced by D_i' connected.

Combining (i) and (ii), D_i' is a connected dominating set with $|D_i'| < |D_i|$, contradicting the minimality of D_i. Hence, every vertex in D_i must be a core point. □

Lemma 4.2. *The union of the ϵ-neighborhoods of the data points in D_i is exactly cluster C_i, i.e., $\bigcup_{p \in D_i} N_\epsilon(p) = C_i$.*

Proof. First, prove $\bigcup_{p \in D_i} N_\epsilon(p) \subseteq C_i$. $\forall q \in \bigcup_{p \in D_i} N_\epsilon(p)$, $\exists\, p \in D_i \subseteq P^{\text{core}}$ (according to Lemma 4.1) such that $\|q - p\| \leq \epsilon$. According to the DBSCAN cluster definition, every point within the ϵ-neighborhood of a core point belongs to the same cluster C_i. Hence, $q \in C_i$.

Then, prove $\bigcup_{p \in D_i} N_\epsilon(p) \supseteq C_i$. $\forall q \in C_i$, if $q \in D_i$, then trivially $q \in \bigcup_{p \in D_i} N_\epsilon(p)$. If $q \notin D_i$, then according to the dominating property of D_i, there exists an edge connecting p and q in G_i^C, which implies $\|p - q\| \leq \epsilon$. Hence $q \in N_\epsilon(p) \subseteq \bigcup_{p \in D_i} N_\epsilon(p)$.

Hence, we conclude that $\bigcup_{p \in D_i} N_\epsilon(p) = C_i$. □

Example 4.2. Figure 1b illustrates the CGraph of the two DBSCAN clusters shown in Figure 1a. It contains two connected components, G_1^C and G_2^C. The MCDS of G_1^C is $D_1 = \{p_2\}$, while that of G_2^C is $D_2 = \{p_3, p_4, p_5\}$. According to Lemma 4.2, the union of the ϵ-neighborhoods of the points in D_1 exactly forms cluster C_1 shown in Figure 1a, and likewise, the union of the ϵ-neighborhoods of the points in D_2 exactly forms cluster C_2 shown in Figure 1a.

Why MCDS? According to the definition of DBSCAN, a cluster is formed by the union of the overlapping ϵ-neighborhoods of core points. To detect DBSCAN clusters, existing algorithms need to find out all core points. In our reformulation, the MCDS D_i of the component corresponding to a cluster C_i is a subset of the core points in C_i (Lemma 4.1), and the cluster can be recovered from D_i (Lemma 4.2). This reduces the number of core points that need to be processed. For example, in Figure 1b, $D_1 = \{p_2\}$ is a subset of the core points of cluster C_1 and $D_2 = \{p_3, p_4, p_5\}$ is a subset of the core points of cluster C_2. As a result, the computational cost associated with core points p_1, p_6, p_7, p_8, and p_9 is eliminated.

4.2 Approximation Algorithm LDBS-KDE

Given the established connection between DBSCAN clusters and the solution to the MCDS of the CGraph, the number of core points to be processed is reduced. However, solving the MCDS problem on the CGraph faces two main challenges. First, the CGraph cannot be materialized until the DBSCAN clusters are explicitly detected, making direct computation of the MCDS infeasible. Second, even when the CGraph is available, the MCDS problem is NP-hard [21], which renders exact computation impractical for large-scale datasets.

To address these challenges, we propose an approximation algorithm, LDBS-KDE, which is motivated by the following observations:

- As the dominating vertices in the MCDS correspond to core points (Lemma 4.1), such vertices generally have *high degrees*, indicating that numerous data points are located near their associated core points.
- If two core points belong to the same cluster, their corresponding dominating vertices are *connected*.
- The dominating vertices are *well separated*, in the sense that the Euclidean distance between their corresponding core points is not too small.

The proposed LDBS-KDE algorithm consists of three phases. First, it selects a set of candidate points using lattice-based density-biased sampling (Section 4.2.1), ensuring that the corresponding dominating vertices are high-degree, connected, and well separated. Next, it identifies core points from these candidates via kernel density estimation (Section 4.2.2), thereby eliminating non-core points. Finally, to recover DBSCAN clusters, it constructs intermediate clusters by grouping the identified core points and then assigns the remaining data points to these clusters to obtain the final results (Section 4.2.3). The overall procedure is summarized in Algorithm 1.

Algorithm 1 LDBS-KDE

Input: A set of n data points P, basis vector length l, sampling ratio ω, ϵ, *MinPts*.
Output: The cluster labels of the data points in P.

1: $S \leftarrow \mathsf{LDBS}(P, l, \omega)$;
2: $S^c \leftarrow \mathsf{CoreKDE}(S, P, \epsilon, \mathit{MinPts})$;
3: $\mathit{Label} \leftarrow \mathsf{RecoverCluster}(S^c, P, \epsilon)$;
4: **return** *Label*;

4.2.1 *Lattice-based Density-Biased Sampling.*

The proposed lattice-based density-biased sampling (LDBS) algorithm (Algorithm 2) selects candidate points with the assistance of lattice points (Definition 4.3).

Definition 4.3 (Lattice point [13]). *Consider the standard orthonormal basis vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_d$ of $\mathbb{R}^d$. We define a lattice $L \subset \mathbb{R}^d$ generated by these basis vectors scaled by a length $l > 0$. A point $\mathbf{r} \in \mathbb{R}^d$ is called a lattice point of L if it can be expressed as an integer linear combination of the scaled basis vectors:*

$$\mathbf{r} = \sum_{i=1}^{d} z_i \cdot (l \cdot \mathbf{e}_i), \quad z_i \in \mathbb{Z}. \tag{4}$$

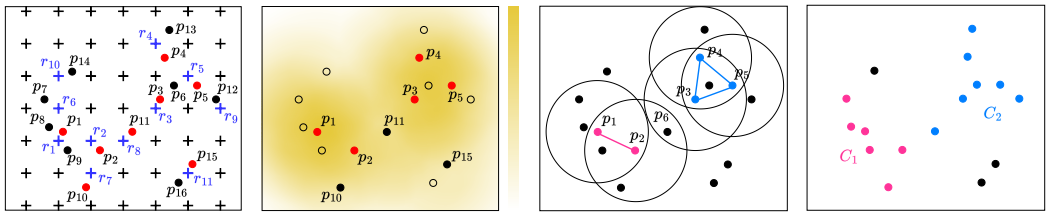
The lattice points serve as a partitioning of the input data space. Each data point is assigned to its nearest lattice point. Lattice points with many assigned data points indicate dense local neighborhoods. Those lattice points corresponding to dense regions are selected, and their nearest data points are chosen as candidates, ensuring that the corresponding vertices have high degrees. The basis vector length l controls the trade-off between the connectivity and separation of the selected candidates.

Algorithm 2 LDBS

Input: A set of n data points P, basis vector length l, sampling ratio ω.
Output: A set of sampled data points S.

1: $L \leftarrow$ generate a lattice using basis vectors with length l;
2: Set $M \leftarrow \emptyset$;
3: **for** each data point p_i in P **do**
4: $\quad r \leftarrow$ the nearest lattice point of p_i according to Equation 5;
5: $\quad d_i = \|p_i - r\|$;
6: $\quad$ **if** $r \notin M$ **then**
7: $\quad\quad$ Create an entry for r;
8: $\quad\quad r.d^* \leftarrow d_i$;
9: $\quad\quad r.\texttt{point} \leftarrow p_i$;
10: $\quad\quad r.\texttt{count} \leftarrow 1$;
11: $\quad\quad$ Add r to M;
12: $\quad$ **else**
13: $\quad\quad r.\texttt{count} \leftarrow r.\texttt{count} + 1$;
14: $\quad\quad$ **if** $d_i < r.d^*$ **then**
15: $\quad\quad\quad r.d^* \leftarrow d_i$;
16: $\quad\quad\quad r.\texttt{point} \leftarrow p_i$;
17: $\quad\quad$ **end if**
18: $\quad$ **end if**
19: **end for**
20: $s \leftarrow n \cdot \omega$;
21: $E \leftarrow$ retrieve the top-s entries by calling QuickSelect(M);
22: **return** $S \leftarrow \{r_j.\texttt{point} \mid r_j \in E\}$;

Specifically, algorithm LDBS (Algorithm 2) takes as input a set of data points P, $|P| = n$, a basis vector length l, and a sampling ratio ω. The sampling ratio ω controls the number of candidate points selected—the larger the ω, the higher the computational cost. A set M is initialized to maintain entries for lattice points (Line 2). Each entry corresponding to a lattice point r stores its



(a) LDBS (b) CoreKDE (c) Grouping core points (d) Clustering result

Fig. 2. The LDBS-KDE algorithm.

nearest data point $r.\texttt{point}$, the distance $r.d^*$ to that data point, and the number $r.\texttt{count}$ of data points whose nearest lattice point is r. For each data point $p_i \in P$, the algorithm determines its nearest lattice point r using Equation 5 and computes the distance d_i between p_i and r (Lines 4–5).

$$r = \sum_{i=1}^{d} z_i \cdot b_i, \quad z_i = \left\lfloor \frac{p \cdot b_i}{l} \right\rceil, \tag{5}$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. If r does not yet exist in M, a new entry for r is created and added to M (Lines 6–11). Otherwise, the count for r is incremented (Line 13), and if p_i is closer than the currently stored point (i.e., $d_i < r.d^*$), the entry is updated to store p_i as the new nearest neighbor (Lines 14–17). Given the input sampling ratio ω, the number of candidate points to be selected is $s = n \cdot \omega$ (Line 20). To efficiently select these s candidates, we apply the QuickSelect algorithm [23] on M to identify the top-s entries based on their $\texttt{count}$ values (Line 21). QuickSelect is an efficient algorithm for finding the s-th smallest or largest element in an unsorted list. It only explores the part of the list that contains the s-th element. We choose QuickSelect because it achieves an average-case time complexity of $O(n)$ and space complexity of $O(1)$, which is better than other selection methods. Finally, the $\texttt{point}$ fields of the top-s entries are returned as the candidates (Line 22).

Figure 2a illustrates the procedure of the LDBS algorithm. Lattice points are denoted by "+". The nearest lattice point of p_1, p_8, and p_9 is r_1, so $r_1.\texttt{count} = 3$. Among these, the nearest data point to r_1 is p_1, hence $r_1.\texttt{point} = p_1$. Given the 16 data points $p_1, p_2, \ldots, p_{16}$ in the figure, the algorithm generates entries for 11 lattice points $r_1, r_2, \ldots, r_{11}$ and stores them in set M. When $s = 8$, the entries of r_1, r_2, r_3, r_4, r_5, r_7, r_8, and r_{11} are selected, returning p_1, p_2, p_3, p_4, p_5, p_{10}, p_{11}, and p_{15} as the candidates, as shown by the red points in the figure.

Parameter l. The basis vector length l influences the selection of candidate points and thus affects the clusters generated in the subsequent steps. The ideal case is that the ϵ-neighborhoods of the candidates cover the true clusters, which requires the candidates to be appropriately spaced. A large l corresponds to a sparse layout of lattice points and leads to the selection of more distant candidate points, potentially causing a true cluster to be split into several smaller ones. Conversely, a small l corresponds to a dense layout of lattice points and results in overly close candidate points. In this case, the algorithm may select redundant candidates, which wastes candidate quotas and may cause some true clusters elsewhere to be missed. We suggest setting l within the range

$$2\epsilon \left(1 - \pi^{\frac{d}{2}} / 4\Gamma\left(\frac{d}{2} + 1\right)\right) \leq l \leq \epsilon, \quad \Gamma(\cdot) \text{ is the Gamma function.} \tag{6}$$

Why $l \leq \epsilon$? Consider the three core points p_1, p_2, and p_3 shown in Figure 3a. According to the definition of DBSCAN clusters, the ϵ-neighborhoods (illustrated as circles) of these three core points belong to the same cluster. When $l \leq \epsilon$, as shown in Figure 3a, the dashed squares surrounding lattice points r_1, r_2, and r_3 indicate the regions where data points take them as their nearest lattice

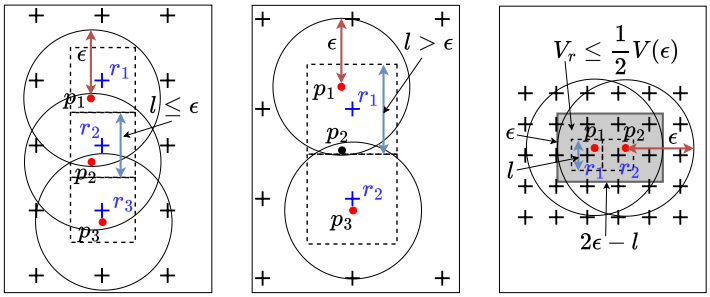


Fig. 3. Illustration of clustering behavior under different values of l.

points. In this case, p_1, p_2, and p_3 are assigned to r_1, r_2, and r_3, respectively, by the LDBS algorithm. These core points are then returned as candidates, ensuring that their ϵ-neighborhoods are merged into the same cluster in subsequent steps. In contrast, when $l > \epsilon$, as shown in Figure 3b, p_1 and p_2 are assigned to r_1, and p_3 is assigned to r_2 by the LDBS algorithm. Since p_1 is closer to r_1 than p_2, p_1 and p_3 are selected as candidates, whereas p_2 is not. Consequently, p_1 and p_3 are not grouped into the same cluster in the subsequent steps because they lie outside each other's ϵ-neighborhoods. As a result, the true cluster formed by the ϵ-neighborhoods of p_1, p_2, and p_3 is incorrectly divided into two smaller clusters.

Why $l \geq 2\epsilon(1 - \pi^{\frac{d}{2}}/4\Gamma(\frac{d}{2}+1))$*?* As discussed earlier, since the dominating vertices are well separated, we need to avoid selecting candidates that are too close to each other. To achieve this, we propose selecting candidates that are well separated in terms of the overlap between their ϵ-neighborhoods. Figure 3c illustrates two core points, p_1 and p_2. For efficiency, we approximate the intersection of their ϵ-neighborhoods (depicted as circles) by a d-dimensional rectangle, whose side length along one dimension is $2\epsilon - l$, approximating the major extent of the intersection region, and ϵ along each of the remaining $d-1$ dimensions. We set the volume $V_r = (2\epsilon - l)\epsilon^{d-1}$ of the d-dimensional rectangle no less than the volume $V(\epsilon) = \pi^{\frac{d}{2}}\epsilon^d/\Gamma(\frac{d}{2}+1)$ of the ϵ-neighborhood of a core point, i.e., $V_r \leq 1/2V(\epsilon)$. Then, we derive $l \geq 2\epsilon(1 - \pi^{\frac{d}{2}}/4\Gamma(\frac{d}{2}+1))$.

4.2.2 *Identifying Core Points using KDE.*

This section introduces the CoreKDE algorithm, which uses kernel density estimation to identify core points from the candidate points generated by the LDBS algorithm. We begin by explaining how to compute the ϵ-neighborhoods of data points using KDE with the uniform kernel (Lemma 4.3).

Lemma 4.3. *Given a data point p, the size of its ϵ-neighborhood is computed as*

$$|N_\epsilon(p)| = \frac{n \cdot \epsilon^d}{\kappa_d} \cdot \mathrm{KDE}_{P,U}(p), \tag{7}$$

where $\mathrm{KDE}_{P,U}(p)$ *denotes the kernel density estimate at p using the uniform kernel over the data set P.*

Proof. The uniform kernel $K_U(\cdot)$ counts the number of points whose distance from p is within h, i.e.,

$$\frac{1}{n \cdot h^d} \sum_{p_i \in P} K_U\left(\frac{\|p - p_i\|}{h}\right) = \frac{\kappa_d}{n \cdot h^d} \sum_{p_i \in P} \begin{cases} 1, & \|p - p_i\| \leq h \\ 0, & \|p - p_i\| > h \end{cases}.$$

Using the uniform kernel in Equation (1), we have:

$$\begin{aligned}\mathsf{KDE}_{P,U}(p) &= \frac{1}{n \cdot h^d} \sum_{p_i \in P} K_U\left(\frac{\|p - p_i\|}{h}\right) \\ &= \frac{\kappa_d}{n \cdot h^d} \sum_{p_i \in P} \begin{cases} 1, & \|p - p_i\| \le h \\ 0, & \|p - p_i\| > h \end{cases} \\ &= \frac{\kappa_d}{n \cdot \epsilon^d} \cdot |N_\epsilon(p)|, \quad \text{by setting} \quad h = \epsilon.\end{aligned}$$

Thus, we have $|N_\epsilon(p)| = \frac{n \cdot \epsilon^d}{\kappa_d} \mathsf{KDE}_{P,U}(p)$. By setting $h = \epsilon$, the ϵ-neighborhoods of data points can be computed using KDE with a uniform kernel. □

However, evaluating $\mathsf{KDE}_{P,U}(p)$ directly is computationally expensive for large datasets. To accelerate this process, we propose estimating the size of the ϵ-neighborhood of p using approximate KDE algorithms with uniform kernel, i.e.,

$$\left|\widehat{N}_\epsilon(p)\right| = \frac{n \cdot \epsilon^d}{\kappa_d} \cdot \widehat{\mathsf{KDE}}_{Q,U}(p), \tag{8}$$

where $\left|\widehat{N}_\epsilon(p)\right|$ approximates $|N_\epsilon(p)|$ and $\widehat{\mathsf{KDE}}_{Q,U}(p)$ denotes an approximate KDE algorithm with the uniform kernel. For efficiency, we adopt a linear-time approximate KDE algorithm `Zrandom+RS` [45]. According to Equation 3, the approximation error of $\left|\widehat{N}_\epsilon(p)\right|$ is:

$$\left|\widehat{N}_\epsilon(p) - N_\epsilon(p)\right| = \frac{n \cdot \epsilon^d}{\kappa_d} \left|\widehat{\mathsf{KDE}}_{Q,U}(p) - \mathsf{KDE}_{P,U}(p)\right| \le \frac{n \cdot \epsilon^d}{\kappa_d} \cdot B. \tag{9}$$

Algorithm 3 presents the pseudo-code of the CoreKDE algorithm. First, the set of core points S^c is initialized as an empty set (Line 1). For each candidate point $p \in S$, it estimates the number of data points in the ϵ-neighborhood of p using Equation 8 (Line 3). If this estimate is at least *MinPts* (Line 4), p is identified as a core point and added to S^c. After all candidates have been processed through the loop (Lines 2–7), the algorithm returns S^c (Line 8).

Algorithm 3 CoreKDE

Input: A set of candidates S, ϵ, *MinPts*.
Output: A set of core points S^c.
1: Set $S^c \leftarrow \emptyset$;
2: **for** each candidate point $p \in S$ **do**
3: $\quad\left|\widehat{N}_\epsilon(p)\right| \leftarrow \frac{n \cdot \epsilon^d}{\kappa_d} \widehat{\mathsf{KDE}}_{Q,U}(p)$;
4: $\quad$**if** $\left|\widehat{N}_\epsilon(p)\right| \ge$ *MinPts* **then**
5: $\quad\quad$Add p to S^c;
6: $\quad$**end if**
7: **end for**
8: **return** S^c;

Figure 2b shows the heatmap of estimated neighborhood sizes for the 8 candidate points p_1, p_2, p_3, p_4, p_5, p_{10}, p_{11}, and p_{15}, returned by the LDBS algorithm. Darker colors indicate larger estimated values. Among these candidates, five are identified as core points by algorithm CoreKDE and highlighted as red points.

4.2.3 *Recovering Clusters.*

This section presents the RecoverCluster algorithm, which reconstructs the DBSCAN clusters of the input point set P based on the core points identified by the CoreKDE algorithm. The pseudo-code is given in Algorithm 4, and the procedure consists of two main steps.

Step 1: Grouping core points. This step groups core points that belong to the same cluster. Given any two core points p and q, if each lies within the other's ϵ-neighborhood, i.e., $p \in N_\epsilon(q)$ and $q \in N_\epsilon(p)$), then they belong to the same cluster according to the DBSCAN definition (cf. Section 3.1). For efficiency, we build a KD-Tree [6] on the core point set S^c (Line 1) and initialize a union-find data structure UF to maintain the groups (Line 2). The algorithm iterates through each core point, retrieving its ϵ-neighborhoods by issuing range queries on the KD-Tree (Lines 3–4). If a neighbor q is found, it is merged with p in the union-find structure (Lines 6–8). Finally, the connected components in UF are identified, and core points within the same component are assigned the same cluster label (Lines 11–16). Consider the core points p_1, p_2, p_3, p_4, p_5 in Figure 2c. Two groups are formed: $g_1 = \{p_1, p_2\}$ and $g_2 = \{p_3, p_4, p_5\}$.

Algorithm 4 RecoverCluster

Input: A set of core points S^c, a set of n data points P, ϵ.
Output: The cluster labels of the points in P.

1: $\mathcal{T} \leftarrow$ build a KD-Tree on S^c;
2: $UF \leftarrow$ initialize a union-find data structure;
3: **for** each point p in S^c **do**
4: $\quad N_\epsilon(p) \leftarrow \mathcal{T}.\textsc{RangeQuery}(p, \epsilon)$;
5: $\quad$ **for** each q in $N_\epsilon(p)$ **do**
6: $\quad\quad$ **if** q has not been processed **then**
7: $\quad\quad\quad UF.\textsc{Union}(p, q)$;
8: $\quad\quad$ **end if**
9: $\quad$ **end for**
10: **end for**
11: $\{C_1, C_2, \dots\} \leftarrow$ identify the connected components in UF;
12: **for** each connected component C_i **do**
13: $\quad$ **for** each point p in C_i **do**
14: $\quad\quad Label(p) \leftarrow i$;
15: $\quad$ **end for**
16: **end for**
17: **for** each $p \in P \setminus S^c$ **do**
18: $\quad N_\epsilon(p) \leftarrow \mathcal{T}.\textsc{RangeQuery}(p, \epsilon)$;
19: $\quad$ **if** $N_\epsilon(p) \neq \emptyset$ **then**
20: $\quad\quad Label(p) \leftarrow Label(q)$, any $q \in N_\epsilon(p)$;
21: $\quad$ **else**
22: $\quad\quad Label(p) \leftarrow -1$; ▷ Noise.
23: $\quad$ **end if**
24: **end for**
25: **return** $Label$;

Step 2: Labeling data points. For each remaining data point $p \in P \setminus S^c$ (Line 17), we issue a range query on the KD-Tree of the core points (Line 18). If there exists a core point $q \in S^c$ within the ϵ-neighborhood of p, then p is assigned the same label as q (Lines 19–20). Otherwise, p is classified

as noise (Lines 21–22). According to the definition of DBSCAN, if a border point falls within the ϵ-neighborhoods of the core points from two different clusters, assigning it to either cluster is considered correct. In Algorithm 4, we assign such border points to the first encountered core point. As illustrated in Figure 2c, the border point p_6 lies within the ϵ-neighborhoods of two core points p_2 and p_3. Consequently, assigning p_6 to either the cluster of p_2 or that of p_3 is valid under the DBSCAN definition.

Figure 2d shows two clusters C_1 and C_2 are reconstructed and the three black points are classified as noise.

4.3 Time Complexity

The LDBS-KDE algorithm consists of Algorithms 2–4. This section analyzes its time complexity.

In Algorithm 2, lines 1-2 run in constant time. Lines 3–19 iterate over the data points in P, where Line 4 performs the computation using Equation 5, resulting in $O(d)$ time per iteration. Therefore, the complexity is $O(n \cdot d)$. Line 20 also takes constant time. Line 21 invokes the QuickSelect algorithm with complexity $O(n_l)$, where $n_l < n$ is the number of entries created for lattice points. Hence, the overall time complexity of Algorithm 2 is $O(n \cdot d)$.

In Algorithm 3, we employ Zrandom+RS [45], which estimates KDE values from a sample set Q of size $|Q| = \lambda$ with approximation error bound B. The algorithm consists of a construction phase with time complexity $O(n + \lambda \log \lambda)$ and a query phase bounded by $O(\lambda)$. In our procedure, line 1 runs in constant time, and lines 2–7 iterate over s sampled points. Among these, line 3 requires $O(\lambda)$ time per iteration, while the comparisons and insertions in lines 4–6 take constant time. Line 8 also requires constant time. Therefore, the overall complexity is $O(n + \lambda \log \lambda + s \cdot \lambda)$.

In Algorithm 4, Line 1 builds a KD-Tree on the core points S^c ($|S^c| \leq s$) in $O(s \log s)$ time, and Line 2 initializes the union-find structure in $O(s)$ time. Lines 3–10 iterate over each core point $p \in S^c$, performing KD-Tree queries and subsequent operations in $O(s)$ per point, for a total of $O(s^2)$. Lines 11–16 assign cluster labels in $O(s)$. Lines 17–24 process non-core points $p \in P \setminus S^c$, with worst-case KD-Tree queries costing $O((n-s)s)$. Overall, the algorithm runs in $O(s \log s + s^2 + (n-s)s) = O(n \cdot s + s \log s)$ time.

Summing the three components, the overall time complexity of the LDBS-KDE algorithm is

$$O\big(n(d+s) + \lambda \log \lambda + s \log s + s \cdot \lambda\big). \tag{10}$$

4.4 Accuracy Analysis

Given a d-dimensional dataset with n data points and DBSCAN parameters ϵ and *MinPts*, we assess the clustering quality of LDBS-KDE by measuring the probabilities that data points from the same (or different) DBSCAN clusters are correctly assigned to the same (or different) clusters by our algorithm.

Lemma 4.4. *Assume that the data points are independently distributed, ϵ is sufficiently small relative to the data space, the estimation error of the chosen KDE randomized algorithm is uniformly distributed in the error interval, and we choose l in the suggested parameter range shown in Equation 6. For any pair of data points belonging to the same DBSCAN cluster that are density-connected through k core points, the probability that they are correctly assigned to the same cluster by* LDBS-KDE *is denoted as P_{same}.*

$$P_{\mathsf{same}} \geq \left(\left(\frac{3}{4} \right)^{d+2} \cdot \frac{l^2}{8\pi\epsilon^2 \cdot (d+1)^2} \right)^{k+1}. \tag{11}$$

Proof. The goal is to bound the probability that two points p and q belonging to the same DBSCAN cluster are correctly assigned to the same cluster by the proposed algorithm.

Step 1: Reduce the problem to local pairwise events.

By the definition of a DBSCAN cluster, if p and q are in the same cluster, there exists a chain of core points $p \leftrightarrow c_1 \leftrightarrow c_2 \leftrightarrow \cdots \leftrightarrow c_k \leftrightarrow q$ such that each adjacent pair is within distance ϵ. Thus, correct clustering of p and q requires that each of the $k+1$ neighboring pairs with distance at most ϵ is correctly merged. Let $P_{\leq\epsilon}$ denote the probability that any such neighboring pair is correctly assigned to the same cluster. Assuming independence, $P_{\texttt{same}} = (P_{\leq\epsilon})^{k+1}$. Therefore, it suffices to derive a lower bound on $P_{\leq\epsilon}$.

Step 2: Case analysis for a neighboring pair (p_1, p_2).

The outcome depends on whether p_1 and/or p_2 are selected as candidates by algorithm LDBS.

Case A: Both points are selected as candidates.

If both p_1 and p_2 are candidates, RecoverCluster then merges them as long as at least one is identified as a core point. By independence, the probability that at least one is correctly identified as a core point is at least 3/4. Thus, $P^A_{\leq\epsilon} \geq \frac{3}{4}$.

Case B: Exactly one point is selected as a candidate.

If one point (e.g., p_1) is selected, then RecoverCluster assigns p_1 and p_2 to the same cluster provided p_1 is correctly identified as a core point by CoreKDE.

Using the uniform-error model for $\widehat{N}_\epsilon(p)$, the probability of correct core identification is at least 1/2. Hence, $P^B_{\leq\epsilon} \geq \frac{1}{2}$.

Case C: Neither point is selected as a candidate.

In this case, p_1 and p_2 can still be clustered together via nearby candidates q_1 and q_2. This requires: each p_i lies within distance ϵ of a corresponding candidate q_i; the candidates q_1 and q_2 are within distance ϵ of each other; at least one of q_1, q_2 is identified as a core point. Then, we derive $P^C_{\leq\epsilon} \geq \left(\frac{3}{4}\right)^{d+2} \cdot \frac{l^2}{8\pi\epsilon^2\cdot(d+1)^2}$.

Step 3: Identify the worst-case probability.

Since $d \geq 1$, the bound from Case C is the smallest among the three cases. Therefore, $P_{\leq\epsilon} = P^C_{\leq\epsilon}$ and Equation 11 is obtained.

Detailed derivations are provided in the supplementary file[1]. □

For intra-cluster data pairs, the lower bounds on the probability of correctness are $P^A_{\leq\epsilon} \geq \frac{3}{4}$ when both points are candidates (Case A) and $P^B_{\leq\epsilon} \geq \frac{1}{2}$ when only one is (Case B). Consequently, decreasing l enhances the likelihood of a data point being selected as a candidate, thereby raising the overall lower bound of the correctness probability. When neither point is selected as a candidate (Case C), the lower bound of the correctness probability decreases as k increases, because an increase in k leads to a longer density-reachable path. Moreover, the probability is affected by the data dimension d. The larger d is, the smaller the lower bound of the probability becomes, indicating that higher dimension exacerbates clustering difficulty.

Lemma 4.5. *Follow the assumption of Lemma 4.4. For any pair of data points p_1 and p_2 from different DBSCAN clusters C_1 and C_2, and let q_1 and q_2 be the closest pair of points from C_1 and C_2, the probability that p_1 and p_2 are correctly assigned to distinct clusters by* LDBS-KDE *is denoted as $P_{\texttt{diff}}$.*

Case A: *If $\|q_1 - q_2\| > \epsilon$, $P_{\texttt{diff}} = 1$.*

Case B: *If $\|q_1 - q_2\| \leq \epsilon$, $P_{\texttt{diff}} \geq \left(\frac{1}{2} + \frac{\kappa_d}{2n\cdot\epsilon^d\cdot B}\right)^{MinPts-1}$.*

Proof. Case A: Well-separated cores.

Let $c_1 \in C_1$ and $c_2 \in C_2$ be the core points such that $p_1 \in N_\epsilon(c_1)$ and $p_2 \in N_\epsilon(c_2)$. Since $\|q_1 - q_2\| > \epsilon$, no core point in C_1 is within ϵ of any core point in C_2. Thus, c_1 and c_2 cannot be merged by RecoverCluster, and neither can p_1, p_2. Hence, $P_{\texttt{diff}} = 1$.

[1]https://github.com/tangjiang777/LDBS-KDE

Case B: Border points.

Both q_1 and q_2 are border points, otherwise they would belong to the same DBSCAN cluster since $\|q_1 - q_2\| \leq \epsilon$. Their shared neighborhood satisfies $|N_\epsilon(q_1) \cap N_\epsilon(q_2)| \leq MinPts - 1$. Correct separation occurs if q_1, q_2 and all points in the intersection are identified as non-core by CoreKDE. Therefore, we have

$$\begin{aligned} P_{\texttt{diff}} &\geq P\left[\widehat{N}_\epsilon(p) < MinPts,\ \ p \in N_\epsilon(q_1) \cap N_\epsilon(q_2)\right] \\ &= \prod_{p \in N_\epsilon(q_1) \cap N_\epsilon(q_2)} \left[\frac{\Delta + (MinPts - N_\epsilon(p))}{2\Delta}\right] \\ &\geq \left(\frac{1}{2} + \frac{1}{2\Delta}\right)^{MinPts-1} = \left(\frac{1}{2} + \frac{\kappa_d}{2n \cdot \epsilon^d \cdot B}\right)^{MinPts-1}. \end{aligned}$$

Detailed derivations are provided in the supplementary file. □

For pairs of data points that should be separated, as long as the clusters they belong to are located far away enough (i.e., the distance between the two closest points of the two clusters is greater than ϵ) they can be perfectly separated in our method. For points within two close clusters, the correct separation is influenced by the *MinPts* and the accuracy of the KDE algorithm.

5 Experiment

The code of our proposed method and the supplementary file are available at https://github.com/tangjiang777/LDBS-KDE.

5.1 Setup

Datasets. We use four real-world datasets, summarized in Table 2. BNG-Aus [3] is a synthetic dataset generated from the raw Australian credit approval data using Bayesian networks. Household [1] consists of electricity consumption records. PAMAP-20 [2] contains multi-modal sensor data collected from wearable devices, for which we applied PCA to retain the top 20 principal dimensions, preserving over 80% of the cumulative variance. SUSY [4] is a high-energy physics dataset comprising simulated particle collision events.

Table 2. Statistics of datasets.

Dataset	#Data points	Dimensionality
BNG-Aus [3]	1,000,000	14
PAMAP-20 [2]	1,667,677	20
Household [1]	2,075,259	7
SUSY [4]	3,916,245	18

Competitors. We compare our approach against the following representative baselines:

- ρ-DBSCAN [18, 19] approximates ϵ-neighborhood queries using a grid index, a technique adopted by many subsequent studies.
- DBSCAN++ [25] forms clusters from sampled data points.
- SNG-DBSCAN [26] constructs clusters using sampled edges in the neighborhood graph.
- BLOCK-DBSCAN [11] identifies clusters at the block level.
- sDBSCAN [43] detects core points via random projections.
- RNN-DBSCAN [7] identifies core points in terms of the reverse nearest neighbor counts.

Although several other approximate DBSCAN variants have been proposed, we exclude them from our evaluation for the following reasons.

KNN-DBSCAN [10] requires as input a precomputed neighborhood graph, which is inconsistent with our experimental setting based on raw point data. NaGB-DBSCAN [30] provides no available implementation and exhibits a quadratic complexity $O(n^2)$, offering no clear theoretical advantage over the methods we compare. KNN-BLOCK DBSCAN [12] is theoretically very close to BLOCK-DBSCAN [11], so we include the latter as a representative method in our comparisons.

Metrics. We compare our approach with competing methods in terms of *elapsed time*. To assess accuracy, we evaluate both our approach and the competitors using the *Adjusted Rand Index* (ARI) [32] and *Normalized Mutual Information* (NMI) [34], with ground-truth clusters derived from the exact DBSCAN algorithm. ARI compares the pairwise agreement between two clustering solutions, with the goal of eliminating the effect of random chance, whereas NMI measures the amount of information shared between the predicted clustering and the true clustering, normalized to avoid scale effects. They capture different aspects of the clustering quality. Employing both metrics together provides a balanced view of accuracy evaluation. All reported metrics are averaged over 10 trials.

Settings. All algorithms are implemented in C++ and executed on the same machine with 4 Intel Xeon E7-4830 CPUs (56 cores, 2.0 GHz) and 2 TB memory and with Ubuntu 16.04.7 LTS.

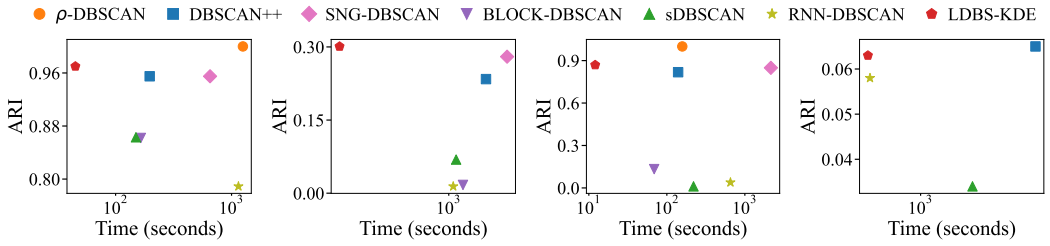
Parameters. We configure the DBSCAN parameters following [16]. We set *MinPts* to $2d + 1$. To determine ϵ, we employ a k-distance graph with $k = MinPts - 1 = 2d$, plotting each point's distance to its k-th nearest neighbor in descending order. The ϵ value is chosen at the elbow of the curve, which marks the transition from dense clusters to sparse noise. All datasets are normalized to the range $(0, 100000)$. Specifically, for the BNG-Aus dataset, *MinPts* and ϵ are set to 29 and 30,000, respectively; for PAMAP-20, to 103 and 40,000; for Household, to 15 and 2,500; and for SUSY, to 37 and 20,000.

For the remaining parameters of the algorithms used in our experiments, the configurations are as follows. For sampling-based algorithms, DBSCAN++ and SNG-DBSCAN, the sampling rate is fixed at 1% by default. In our algorithm, parameter l is set following the recommended range in Equation 6. Specifically, we conducted a binary search within this interval for each dataset and selected the value that yielded good performance. l is set to ϵ for the BNG-Aus dataset, ϵ for PAMAP-20, $2\epsilon/3$ for Household, and $5\epsilon/6$ for SUSY. Parameter ω denotes the user-specified sampling ratio. Based on our experimental observations, a sampling rate between 1% and 3% already delivers strong performance, and larger values are unnecessary. The approximation error bound B is customized for each dataset such that the number of sampled points in the approximate KDE algorithm accounts for 5% of the total data. For the other algorithms, we first adopt the parameter settings recommended in their original papers and adjust them when necessary to ensure a runtime comparable to that of the sampling-based algorithms. For ρ-DBSCAN, we set $\rho = 0.1$; for sDBSCAN, we set $D = 256$, $topK = 10$, and $topM = MinPts$; for RNN-DBSCAN, we set $k = 64$ for the BNG-Aus dataset, 47 for PAMAP-20, 23 for Household, and 6 for SUSY. BLOCK-DBSCAN requires no parameter tuning.

5.2 Results

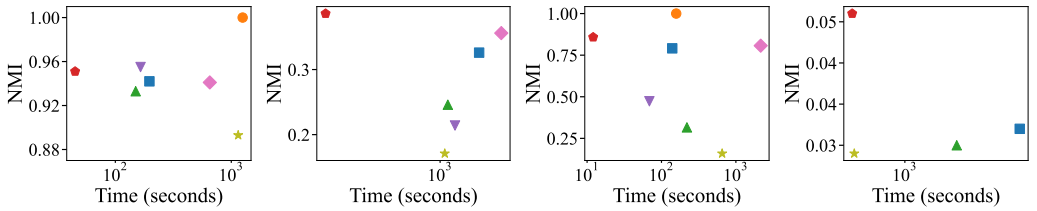
5.2.1 Accuracy vs. Efficiency. Figure 4 compares our algorithm with competitors on four datasets in terms of ARI versus elapsed time. Similarly, Figure 5 presents the comparison in terms of NMI and elapsed time.

Accuracy. Our algorithm, LDBS-KDE, outperforms BLOCK-DBSCAN, sDBSCAN, SNG-DBSCAN, and DBSCAN++ in terms of both ARI and NMI in most cases, as it leverages the advantages of the MCDS formulation, ensuring that the identified core points maintain the correct cluster structure effectively. In most cases, RNN-DBSCAN, BLOCK-DBSCAN, and sDBSCAN perform worse than



(a) BNG-Aus (b) PAMAP-20 (c) Household (d) SUSY

Fig. 4. ARI vs. elapsed time.



(a) BNG-Aus (b) PAMAP-20 (c) Household (d) SUSY

Fig. 5. NMI vs. elapsed time.

the other algorithms. This is because BLOCK-DBSCAN divides the data into blocks, so that points near the boundaries of blocks might not be assigned to the correct cluster. sDBSCAN uses a limited number of random projection vectors for efficiency, but it introduces significant errors in neighborhood estimation, leading to the misidentification of core points. RNN-DBSCAN replaces the geometric density definition (ϵ-ball) with Reverse Nearest Neighbor (RNN) counts. Relying on a single parameter k makes it difficult to capture precise structural details globally. Furthermore, it uses an approximate nearest neighbor algorithm to improve efficiency, but introduces errors that degrade clustering quality. SNG-DBSCAN and DBSCAN++ are generally slightly less effective than our algorithm, though in some cases, their performance is comparable. This is because they rely on simple random sampling, which often oversamples dense regions while overlooking sparse areas, resulting in less accurate clustering results. Although ρ-DBSCAN performs slightly better than our algorithm, it fails to return any results on the PAMAP-20 and SUSY datasets due to its high time complexity.

Efficiency. With respect to elapsed time, LDBS-KDE demonstrates superior efficiency compared to all competitors across all datasets. ρ-DBSCAN is slow because its computational complexity grows exponentially with dimensionality, and it failed to produce results within three hours on the high-dimensional datasets PAMAP-20 and SUSY. The sampling-based algorithms, DBSCAN++ and SNG-DBSCAN, perform clustering over the entire dataset, leading to high computational costs; SNG-DBSCAN did not return results within three hours on the SUSY dataset. In contrast, our algorithm conducts clustering only on the sampled data, resulting in substantially lower computation time. sDBSCAN must check the core-point condition for every data point, which incurs significant computational overhead. BLOCK-DBSCAN relies on a Cover Tree to perform range queries for nearly all data points, and the efficiency of this operation deteriorates rapidly in high-dimensional spaces; it also failed to return results within three hours on the SUSY dataset. RNN-DBSCAN must calculate reverse nearest neighbors for all points, it remains computationally expensive even when leveraging approximate nearest neighbor search techniques. On the BNG-Aus dataset, LDBS-KDE runs approximately an order of magnitude faster than ρ-DBSCAN, 3–5× faster than DBSCAN++, BLOCK-DBSCAN, and sDBSCAN, and 15× faster than SNG-DBSCAN. On the PAMAP-20 dataset,

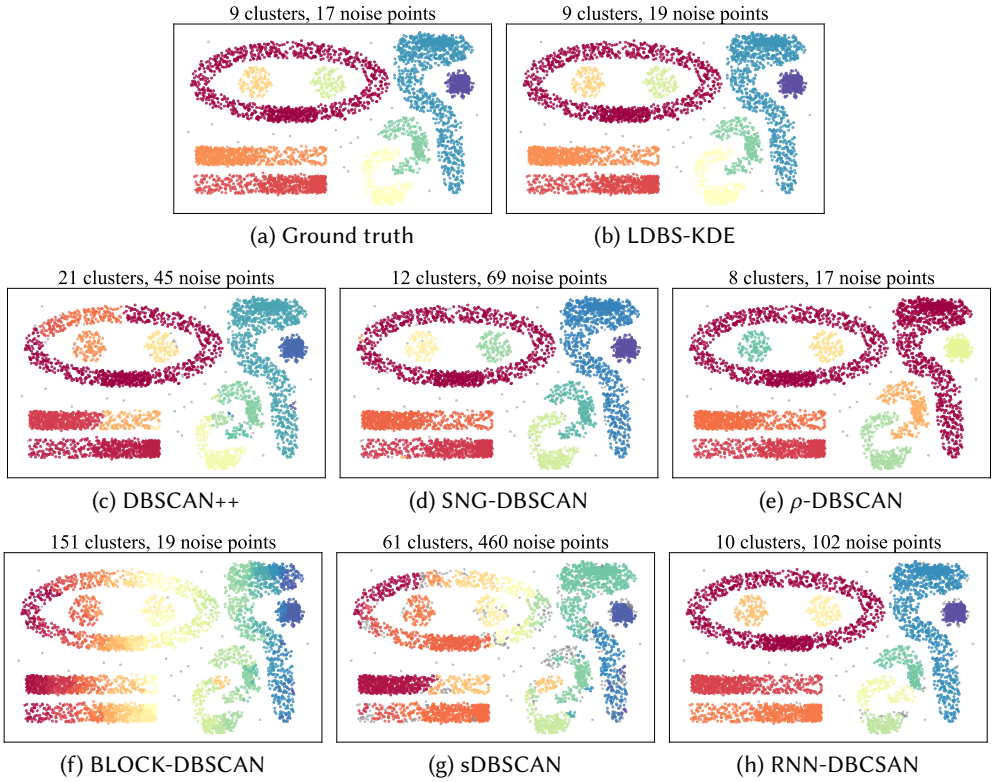
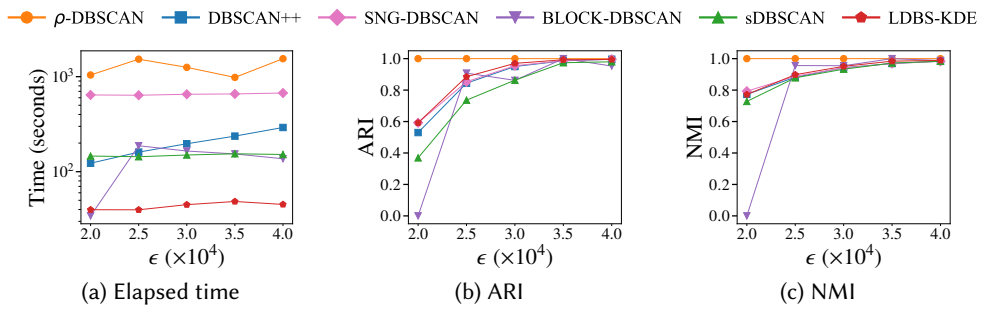


Fig. 6. Case study.

LDBS-KDE achieves up to a 2× speedup over BLOCK-DBSCAN and sDBSCAN, and is an order of magnitude faster than DBSCAN++ and SNG-DBSCAN. On the Household dataset, LDBS-KDE runs several orders of magnitude faster than ρ-DBSCAN, DBSCAN++, SNG-DBSCAN, and sDBSCAN, and is 5× faster than BLOCK-DBSCAN. For the SUSY dataset, it achieves a 2–4× speedup over DBSCAN++ and sDBSCAN.

Summary. Across all datasets, LDBS-KDE demonstrates consistently high ARI and NMI values while maintaining the lowest computational cost. This confirms that our approach achieves an effective trade-off between clustering quality and efficiency, providing competitive and reliable performance across a variety of datasets.

5.2.2 Case Study. To investigate the clustering results of our algorithm and the competitors, we conduct a case study on a synthetic dataset containing clusters with irregular shapes, varying sizes and densities. Figure 6a visualizes the 9 clusters and 17 noise points in the dataset. Parameters $\epsilon = 5$ and $MinPts = 15$ are set identically for all algorithms. In our algorithm, LDBS-KDE, we set $l = 9$, $B = 0.05$, and the number of candidates in Algorithm 2 is chosen to match the number of entries in M, corresponding to a sampling ratio of $\omega = 31.5\%$. For fair comparison, the sampling ratios for DBSCAN++ and SNG-DBSCAN are set to 35%. For sDBSCAN and ρ-DBSCAN, due to the significant impact of their parameters on runtime, we adjusted them to ensure their runtimes are comparable to ours. This setting ensures that their parameters are optimized for clustering quality without causing their runtimes to significantly exceed that of our algorithm. Specifically, for sDBSCAN, we set $D = 256$, $topK = 10$, and $topM = 5$, while for ρ-DBSCAN, we set $\rho = 0.2$. BLOCK-DBSCAN requires no parameter tuning. As RNN-DBSCAN requires only a single hyperparameter k (independent of ϵ



(a) Elapsed time (b) ARI (c) NMI

Fig. 7. Varying ϵ on the BNG-Aus dataset.

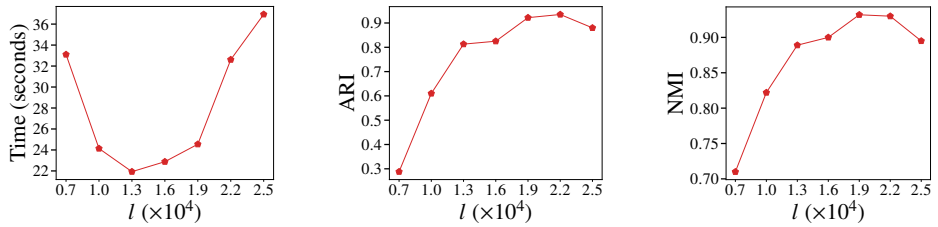
and *MinPts*), we determined its value using the frequency-based heuristic recommended by the authors, setting $k = 26$.

Figure 6 illustrates the clustering results of our algorithm, LDBS-KDE, alongside the competitors. Our algorithm outperforms all competitors, achieving the highest clustering quality by correctly identifying all 9 clusters and misclassifying only 2 noise points. In contrast, DBSCAN++, SNG-DBSCAN, BLOCK-DBSCAN, sDBSCAN, and RNN-DBSCAN each split several true clusters into multiple smaller ones and misclassify a larger number of data points as noise. ρ-DBSCAN, on the other hand, merges two true clusters into a single cluster.

5.2.3 Effect of parameters. This section reports the performance of different algorithms under different parameter settings.

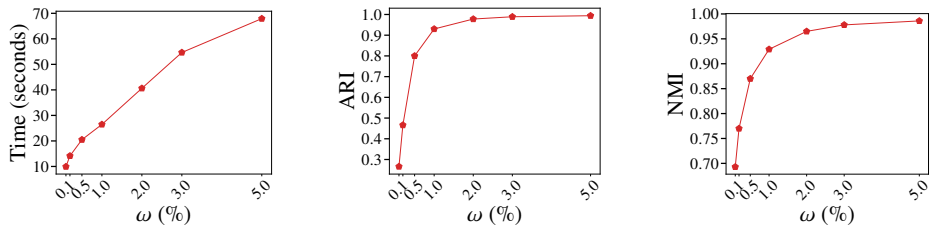
Varying ϵ. Figure 7 shows the elapsed time, ARI, and NMI of our algorithm and the competitors when varying the DBSCAN parameter ϵ on the BNG-Aus dataset. Since RNN-DBSCAN does not involve the parameter ϵ, it is excluded from this set of experiments. In terms of elapsed time, LDBS-KDE achieves low computational cost across all values of ϵ. The elapsed time of DBSCAN++ increases slightly as ϵ increases due to the increased cost of neighborhood queries, while BLOCK-DBSCAN terminates early when no clusters are formed, yielding short elapsed time for small ϵ. The elapsed time of ρ-DBSCAN, SNG-DBSCAN, and sDBSCAN remain relatively stable with respect to ϵ. Regarding accuracy as measured by ARI and NMI, all algorithms except ρ-DBSCAN exhibit an upward trend with increasing ϵ. For the sampling-based algorithms LDBS-KDE, DBSCAN++, and SNG-DBSCAN, as ϵ increases, neighborhoods cover more data points at a fixed sampling ratio, thereby improving clustering accuracy. In sDBSCAN, which identifies neighborhood points via random vector projections, smaller ϵ values correspond to finer local structures that require more projection vectors; with a fixed number of vectors, accuracy decreases. BLOCK-DBSCAN partitions the space into blocks of size $\epsilon/2$ to detect core blocks. When ϵ is small, these blocks become overly fine, preventing the detection of internal core blocks, so meaningful clusters fail to form and both ARI and NMI approach zero. In contrast, ρ-DBSCAN remains insensitive to ϵ due to its strict density and connectivity criteria.

Varying l. Figure 8 shows the elapsed time, ARI, and NMI of our algorithm on the BNG-Aus dataset when varying parameter l. As l increases from 0.7×10^4 to 1.3×10^4, the elapsed time decreases significantly. When l is small (e.g., 0.7×10^4), the lattice points are relatively dense, and the selected candidate points cover fewer data points. In the subsequent steps, the KD-tree used for nearest-neighbor search encounters many queries in its worst-case scenario, resulting in high computational cost. As l increases, such worst-case queries become less frequent, reducing the overall cost. Conversely, when l increases from 1.3×10^4 to 2.5×10^4, the lattice points become sparser, and the candidate points cover more data points. This involves more points in the clustering process, thereby increasing the computational cost. Both ARI and NMI increase as l grows, reaching



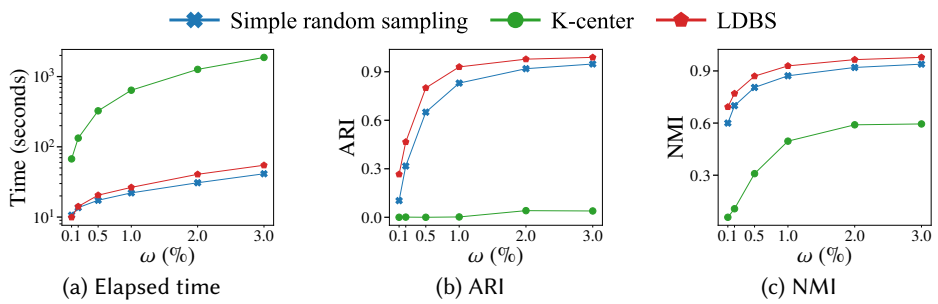
(a) Elapsed time (b) ARI (c) NMI

Fig. 8. Varying l in LDBS-KDE on the BNG-Aus dataset.



(a) Elapsed time (b) ARI (c) NMI

Fig. 9. Varying ω in LDBS-KDE on the BNG-Aus dataset.



(a) Elapsed time (b) ARI (c) NMI

Fig. 10. Effect of sampling strategy in LDBS-KDE on the BNG-Aus dataset.

their peak when l varies from 1.9×10^4 to 2.2×10^4, and then decline slightly. A moderate increase in l allows the selected candidate points to cover more data points, improving clustering effectiveness. However, if l becomes too large, the distance between lattice points grows excessively, causing the candidate points to lose connectivity and making it difficult to form valid clusters.

Varying ω. Figure 9 shows the elapsed time, ARI, and NMI of our algorithm on the BNG-Aus dataset when varying parameter ω. As the sampling ratio ω increases from 0.1% to 5.0%, the elapsed time of our algorithm increases from approximately 10 seconds to 70 seconds. This is primarily because ω determines the number of candidates selected. A large ω requires more candidates to be processed, leading to more data points being involved in the clustering process. When ω is small (e.g., below 0.5%), clustering accuracy is low. As ω increases to around 1.5%, both ARI and NMI rise and then stabilize, reaching near-optimal performance. This indicates that a higher sampling ratio allows the candidate points to better preserve the underlying clustering structure, thereby improving accuracy. However, when $\omega \geq 2\%$, accuracy shows little further improvement, suggesting that these additional candidates contribute minimally. This observation supports the motivation behind the MCDS formulation, which posits that only a subset of core points is sufficient to recover the clusters.

Effect of sampling strategy. Figure 10 illustrates the effect of different sampling strategies in LDBS-KDE in terms of elapsed time, ARI, and NMI, including *simple random sampling*, *K-center* [25], and our proposed algorithm LDBS. In terms of elapsed time, *K-center* incurs a high computational cost that increases linearly with sampling ratio ω. This is because it repeatedly computes distances between each data point and the current cluster centers, resulting in a time complexity of $O(n \cdot k)$. In contrast, both *simple random sampling* and LDBS are significantly more efficient and comparable in elapsed time. Regarding clustering accuracy, *K-center* performs the worst across all sampling ratios. While it aims to maximize spatial coverage, a low sampling ratio leads to sparsely distributed points that lack local connectivity, making it difficult to capture cluster structures accurately. Given a fixed sampling ratio, *simple random sampling* may oversample dense regions, leading to less accurate clustering results. In contrast, LDBS consistently achieves the best performance in terms of ARI and NMI by selecting candidates from dense regions while maintaining sufficient coverage. Overall, LDBS strikes an excellent balance between efficiency and accuracy, delivering superior clustering performance at a substantially lower computational cost compared with the other two sampling strategies.

6 Conclusion

We reformulate the DBSCAN clustering problem as a Minimum Connected Dominating Set (MCDS) problem to reduce the number of core points that need to be processed, thereby lowering the computational cost. Given the NP-hardness of the MCDS problem, we propose an approximate algorithm, LDBS-KDE, which efficiently computes accurate clusters for large-scale datasets. Specifically, LDBS-KDE first selects candidate points using density-biased sampling assisted by lattice points, then identifies core points from these candidates via kernel density estimation, forms intermediate clusters by grouping the identified core points, and finally assigns the remaining data points to their nearest core points. Extensive experiments confirm that LDBS-KDE achieves accurate clustering at a significantly reduced computational cost.

Future research directions include extending LDBS-KDE to streaming settings and exploring its applicability to other clustering frameworks.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants 62372308, 62572326, and 23IAA00610, and Scientific Foundation for Youth Scholars of Shenzhen University.

References

[1] 2012. *Household in UCI.* https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption
[2] 2012. *PAMAP2 Physical Activity Monitoring.* https://archive.ics.uci.edu/dataset/231/pamap2+physical+activity+monitoring
[3] 2014. *BNG (Australian) Open Data.* https://www.openml.org/search?type=data&status=active&id=1205&sort=runs
[4] 2014. *SUSY.* https://archive.ics.uci.edu/dataset/279/susy
[5] Arturs Backurs, Piotr Indyk, and Tal Wagner. 2019. Space and Time Efficient Kernel Density Estimation in High Dimensions. In *NIPS*. 15773–15782.
[6] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.
[7] Avory Bryant and Krzysztof J. Cios. 2018. RNN-DBSCAN: A Density-Based Clustering Algorithm Using Reverse Nearest Neighbor Density Estimates. *IEEE Trans. Knowl. Data Eng.* 30, 6 (2018), 1109–1121.
[8] Moses Charikar, Michael Kapralov, Navid Nouri, and Paris Siminelakis. 2020. Kernel Density Estimation through Density Constrained Near Neighbor Search. In *FOCS*. 172–183.
[9] Moses Charikar and Paris Siminelakis. 2017. Hashing-Based-Estimators for Kernel Density in High Dimensions. In *FOCS*. 1032–1043.

[10] Youguang Chen, William Ruys, and George Biros. 2025. KNN-DBSCAN: a DBSCAN in high dimensions. *ACM Trans. Parallel Comput.* 12, 1 (2025), 3:1–3:27.

[11] Yewang Chen, Lida Zhou, Nizar Bouguila, Cheng Wang, Yi Chen, and Jixiang Du. 2021. BLOCK-DBSCAN: Fast clustering for large scale data. *Pattern Recognit.* 109 (2021), 107624.

[12] Yewang Chen, Lida Zhou, Songwen Pei, Zhiwen Yu, Yi Chen, Xin Liu, Jixiang Du, and Naixue Xiong. 2021. KNN-BLOCK DBSCAN: Fast Clustering for Large-Scale Data. *IEEE Trans. Syst. Man Cybern. Syst.* 51, 6 (2021), 3939–3953.

[13] John H. Conway and Neil J. A. Sloane. 1988. *Sphere Packings, Lattices and Groups*. Vol. 290. Springer.

[14] Chao Deng, Jinwei Song, Ruizhi Sun, Saihua Cai, and Yinxue Shi. 2018. GRIDEN: An effective grid-based and density-based spatial clustering algorithm to support parallel computing. *Pattern Recognit. Lett.* 109 (2018), 81–88.

[15] Gary Doran, David R. Thompson, and Tara A. Estlin. 2016. Precision Instrument Targeting via Image Registration for the Mars 2020 Rover. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 3352–3358.

[16] 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 *KDD*. 226–231.

[17] Joseph Alejandro Gallego, Juan F. Osorio, and Fabio A. González. 2022. Fast Kernel Density Estimation with Density Matrices and Random Fourier Features. In *IBERAMIA*. 160–172.

[18] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *SIGMOD*. 519–530.

[19] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Trans. Database Syst.* 42 (2017), 14:1–14:45.

[20] Teofilo F. Gonzalez. 1985. Clustering to Minimize the Maximum Intercluster Distance. *Theor. Comput. Sci.* 38 (1985), 293–306.

[21] Sudipto Guha and Samir Khuller. 1998. Approximation Algorithms for Connected Dominating Sets. *Algorithmica* 20, 4 (1998), 374–387.

[22] Felix Hausdorff. 1914. *Grundzüge der Mengenlehre*. Leipzig Viet.

[23] C. A. R. Hoare. 1961. Algorithm 65: find. *Commun. ACM* 4, 7 (1961), 321–322.

[24] Yourong Huang, Zhu Xiao, Xiaoyou Yu, Dong Wang, Vincent Havyarimana, and Jing Bai. 2019. Road Network Construction with Complex Intersections Based on Sparsely Sampled Private Car Trajectory Data. *ACM Trans. Knowl. Discov. Data* 13 (2019), 35:1–35:28.

[25] Jennifer Jang and Heinrich Jiang. 2019. DBSCAN++: Towards fast and scalable density clustering. In *ICML*. 3019–3029.

[26] Heinrich Jiang, Jennifer Jang, and Jakub Lacki. 2020. Faster DBSCAN via subsampled similarity queries. In *NeurIPS*. 22407–22419.

[27] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* 7, 3 (2021), 535–547.

[28] Matti Karppa, Martin Aumüller, and Rasmus Pagh. 2022. DEANN: Speeding up Kernel-Density Estimation using Approximate Nearest Neighbor Search. In *AISTATS*. 3108–3137.

[29] Yang Li, Rongshun Juan, Yatao Zhou, Tianshu Wang, Zezhong Li, Wei Guo, and Zhongke Gao. 2025. AD-RRT*: An RRT*-based global path planning approach for underwater gliders with alpha shapes and DBSCAN. *Expert Syst. Appl.* 291 (2025), 128219.

[30] Ranliang Luo, Tianshuo Li, Rui Pu, Juntao Yang, Dongming Tang, and Lijun Yang. 2025. NaGB-DBSCAN: An improved DBSCAN clustering algorithm by natural neighbor and granular-ball. *Inf. Sci.* 719 (2025), 122445.

[31] Shaaban Mahran and Khaled Mahar. 2008. Using grid for accelerating density-based clustering. In *IEEE CIT*. 35–40.

[32] Xuan Vinh Nguyen, Julien Epps, and James Bailey. 2009. Information theoretic measures for clusterings comparison: is a correction for chance necessary?. In *ICML*. 1073–1080.

[33] Emanuel Parzen. 1962. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics* 33, 3 (1962), 1065–1076.

[34] Darius Pfitzner, Richard Leibbrandt, and David M. W. Powers. 2009. Characterization and evaluation of similarity measures for pairs of clusterings. *Knowl. Inf. Syst.* 19, 3 (2009), 361–394.

[35] Jeff M. Phillips. 2013. ϵ-Samples for Kernels. In *SODA*. 1622–1632.

[36] Jeff M. Phillips and Wai Ming Tai. 2018. Improved Coresets for Kernel Density Estimates. In *SODA*. 2718–2727.

[37] Jeff M. Phillips and Wai Ming Tai. 2020. Near-Optimal Coresets of Kernel Density Estimates. *Discret. Comput. Geom.* 63 (2020), 867–887.

[38] Ali Rahimi and Benjamin Recht. 2007. Random Features for Large-Scale Kernel Machines. In *NIPS*. 1177–1184.

[39] Ye Shiqiu and Zhu Qingsheng. 2019. DBSCAN clustering algorithm based on locality sensitive hashing. *Journal of Physics: Conference Series* 1314, 1 (2019), 012177.

[40] Bernard W. Silverman. 1986. *Density Estimation for Statistics and Data Analysis*. Springer.

[41] Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Alexander Levis. 2019. Rehashing Kernel Evaluation in High Dimensions. In *ICML*. 5789–5798.
[42] Wai Ming Tai. 2022. Optimal Coreset for Gaussian Kernel Density Estimation. In *SoCG*. 63:1–63:15.
[43] Haochuan Xu and Ninh Pham. 2024. Scalable DBSCAN with Random Projections. In *NIPS*. 27978–28008.
[44] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*. 242–253.
[45] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. 2013. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*. 433–444.

Received October 2025; revised January 2026; accepted February 2026