DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation*

Junhao Gan Yufei Tao
Chinese University of Hong Kong
New Territories, Hong Kong
{jhgan, taoyf}@cse.cuhk.edu.hk

ABSTRACT

DBSCAN is a popular method for clustering multi-dimensional objects. Just as notable as the method's vast success is the research community's quest for its efficient computation. The original KDD'96 paper claimed an algorithm with $O(n \log n)$ running time, where n is the number of objects. Unfortunately, this is a mis-claim; and that algorithm actually requires $O(n^2)$ time. There has been a fix in 2D space, where a genuine $O(n \log n)$-time algorithm has been found. Looking for a fix for dimensionality $d \geq 3$ is currently an important open problem.

In this paper, we prove that for $d \geq 3$, the DBSCAN problem requires $\Omega(n^{4/3})$ time to solve, unless very significant breakthroughs—ones widely believed to be impossible—could be made in theoretical computer science. This (i) explains why the community's search for fixing the aforementioned mis-claim has been futile for $d \geq 3$, and (ii) indicates (sadly) that *all* DBSCAN algorithms must be intolerably slow even on moderately large n in practice. Surprisingly, we show that the running time can be dramatically brought down to $O(n)$ in expectation *regardless of the dimensionality* d, as soon as slight inaccuracy in the clustering results is permitted. We formalize our findings into the new notion of *ρ-approximate DBSCAN*, which we believe should replace DBSCAN on big data due to the latter's computational intractability.

Categories and Subject Descriptors
H.3.3 **[Information search and retrieval]:** Clustering

Keywords
DBSCAN, Density-Based Clustering, Algorithm

1. INTRODUCTION

Density-based clustering is one of the most fundamental topics in data mining. Given a set P of n points in d-dimensional space $\mathbb{R}^d$, the objective is to group the points of P into subsets—called *clusters*—such that any two clusters are separated by "sparse regions". Figure 1 shows two classic examples taken from [10]:

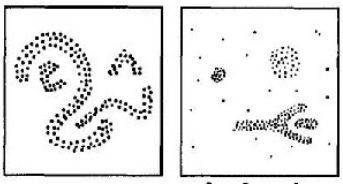


Figure 1: Examples of density-based clustering from [10]

the left one contains 4 snake-shaped clusters, while the right one contains 3 clusters together with some noise. The main advantage of density-based clustering (over methods such as *k-means*) is its capability of discovering clusters with arbitrary shapes (while k-means typically returns ball-like clusters).

Density-based clustering can be achieved using a variety of approaches, which differ mainly in their (i) definitions of "dense/sparse regions", and (ii) criteria of how dense regions should be connected to form clusters. In this paper, we concentrate on DBSCAN, which is an elegant approach invented by Ester, Kriegel, Sander, and Xu [10], and received the test-of-time award in KDD'14. DBSCAN characterizes "density/sparsity" by resorting to two parameters:

- ϵ: a positive real value;
- $MinPts$: a small positive constant integer.

Let $B(p, \epsilon)$ be the d-dimensional ball centered at point p with radius ϵ, where the distance metric is Euclidean distance. $B(p, \epsilon)$ is "dense" if it covers at least $MinPts$ points of P.

DBSCAN forms clusters based on the following rationale. *If $B(p, \epsilon)$ is dense, all the points in $B(p, \epsilon)$ should be added to the same cluster as p.* This creates a "chained effect": whenever a new point p' with a dense $B(p', \epsilon)$ is added to the cluster of p, all the points in $B(p', \epsilon)$ should also join the same cluster. The cluster of p continues to grow in this manner to the effect's fullest extent.

1.1 A Mis-Claim of 17 Years

In their seminal paper [10], Ester et al. claimed that their DBSCAN algorithm terminates in $O(n \log n)$ time. This turns out to be a mis-claim: as pointed out by Gunawan [11] recently, the algorithm of [10] actually runs in $O(n^2)$ worst case time, regardless of the parameters ϵ and $MinPts$.[1]

The above error, unfortunately, has permeated deeply into the database and data mining fields. For example, the $O(n \log n)$

*This work was supported in part by projects GRF 4168/13 and GRF 142072/14 from HKRGC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright © 2015 ACM 978-1-4503-2758-9/15/05 ...$15.00.
Http://dx.doi.org/10.1145/2723372.2737792.

[1]This is in fact quite obvious in retrospect. The algorithm of [10] performs n *range queries*, each of which reports all the points within distance ϵ from a data point. When the points of P are all within distance ϵ from each other, the total time of all those queries is already $O(n^2)$.

"performance bound" is explicitly stated in the Wikipedia page of DBSCAN[2], major textbooks [12, 20, 25], and a long string of papers [2, 5, 7, 13, 16, 19, 24, 28, 29, 30] (mentioning just 10 of them). Very unfortunately, adverse consequence has ensued: several papers [14, 21, 23] have utilized the $O(n \log n)$ "claim" as a building brick to derive new "results", which have thus all been invalidated the moment the error was found (specifically, the affected results lie in Sec D.1 of [14], Sec 3.2 of [21], and Sec 5.2 of [23]).

In [11], Gunawan also showed that *all* of the subsequently improved versions of the original DBSCAN algorithm either do not compute the precise DBSCAN result (e.g., see [6, 15, 26]), or still suffer from $O(n^2)$ running time [17]. As a partial remedy, Gunawan developed a new 2D algorithm which truly runs in $O(n \log n)$ time, but left the $d \geq 3$ case as an open problem.

The mis-claim in [10] and its 2D remedy [11] leave behind two intriguing questions:

1. For $d \geq 3$, is it possible to fix the error in [10] by designing an algorithm that genuinely has $O(n \log n)$ time complexity? To make things easier, is it possible to achieve time complexity $O(n \log^c n)$ even for some *very large* constant c?
2. If the answer to the previous question is *no*, it means that DBSCAN in $d \geq 3$ is computationally *intractable* in practice even on moderately large n. What should we do if we still want to apply this method to cluster a large dataset? This question becomes increasingly urgent nowadays with the arrival of big data.

1.2 Our Contributions

This paper makes three contributions. First, we prove that the DBSCAN problem requires $\Omega(n^{4/3})$ time to solve in $d \geq 3$, unless *very* significant breakthroughs (ones widely believed to be impossible) can be made in theoretical computer science. Note that $n^{4/3}$ is *arbitrarily* larger than $n \log^c n$, regardless of constant c.

Second, we introduce a new concept called *ρ-approximate DBSCAN* as an alternative to DBSCAN on large datasets of $d \geq 3$. ρ-approximate DBSCAN comes with very strong assurances in both *quality* and *efficiency*. For quality, its clustering result is guaranteed to be "sandwiched" between the results of DBSCAN obtained with parameters $(\epsilon, MinPts)$ and $(\epsilon(1+\rho), MinPts)$, respectively. This is very desired in practice, because it is well-known [2] that there is a comfortable range of ϵ that will yield good DBSCAN clusters. For efficiency, we prove that ρ-approximate DBSCAN can be solved in linear $O(n)$ expected time, *for any ϵ, arbitrarily small constant ρ, and in any fixed dimensionality d!* It is rather surprising that such a small sacrifice of accuracy can bring this tremendous gain in running time.

Third, we perform DBSCAN experiments on datasets significantly larger than those used in all the previous experiments to our awareness. The experiments reveal that, as predicted by theory, *none* of the exact DBSCAN algorithms has acceptable running time even in 3D (which perhaps explains why the previous evaluation was done only on small datasets). In contrast, our new ρ-approximate DBSCAN algorithm exhibits graceful scalability with respect to all parameters, and outperforms even the fastest exact algorithm by a factor up to *three orders of magnitude*.

1.3 Paper Organization

Section 2 reviews the previous work related to ours. Section 3 provides theoretical evidence on the computational intractability of DBSCAN in practice, and discusses what practitioners can do if they insist on solving the problem exactly. Section 4 proposes ρ-approximate DBSCAN, elaborates on our algorithm, and establishes its quality and efficiency guarantees. Section 5 evaluates the exact and approximation algorithms with extensive experimentation. Finally, Section 6 concludes the paper with a summary of findings.

[2] *http://en.wikipedia.org/wiki/DBSCAN*

2. RELATED WORK

Section 2.1 reviews the DBSCAN definitions as set out by Ester et al. in [10]. Section 2.2 describes the 2D algorithm in [11] that solves the problem genuinely in $O(n \log n)$ time. Section 2.3 points out several results from computational geometry which will be needed to prove the intractability of DBSCAN later.

2.1 Definitions

As before, let P be a set of n points in d-dimensional space $\mathbb{R}^d$. Given two points $p, q \in \mathbb{R}^d$, we denote by $dist(p, q)$ the Euclidean distance between p and q. Denote by $B(p, r)$ the ball centered at a point $p \in \mathbb{R}^d$ with radius r. Remember that DBSCAN takes two parameters: ϵ and $MinPts$.

DEFINITION 1. *A point $p \in P$ is a* **core point** *if $B(p, \epsilon)$ covers at least $MinPts$ points of P (including p itself).*

If p is not a core point, it is said to be a *non-core point*. To illustrate, suppose that P is the set of points in Figure 2, where $MinPts = 4$ and the two circles have radius ϵ. Core points are shown in black, and non-core points in white.

DEFINITION 2. *A point $q \in P$ is* **density-reachable** *from $p \in P$ if there is a sequence of points $p_1, p_2, ..., p_t \in P$ (for some integer $t \geq 2$) such that:*

- *$p_1 = p$ and $p_t = q$*
- *$p_1, p_2, ..., p_{t-1}$ are core points*
- *$p_{i+1} \in B(p_i, \epsilon)$ for each $i \in [1, t-1]$.*

Note that points p and q do *not* need to be different. In Figure 2, for example, o_1 is density-reachable from itself; o_{10} is density-reachable from o_1 and from o_3 (through the sequence o_3, o_2, o_1, o_{10}). On the other hand, o_{11} is *not* density-reachable from o_{10} (recall that o_{10} is not a core point).

DEFINITION 3. *A* **cluster** *C is a non-empty subset of P such that:*

- *(Maximality) If a core point $p \in C$, then all the points density-reachable from p also belong to C.*

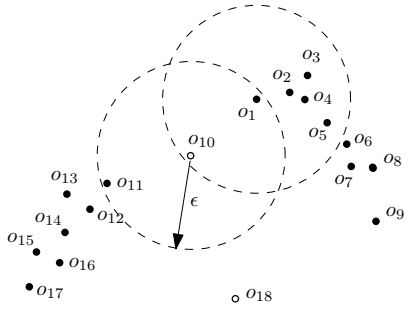
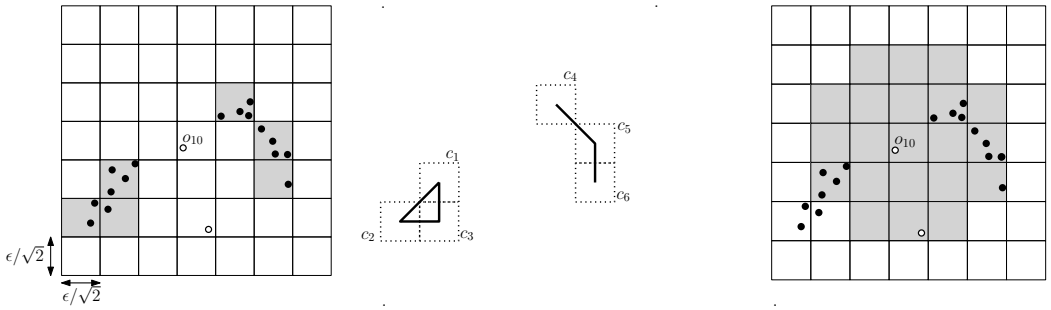


Figure 2: An example dataset (the two circles have radius ϵ; $MinPts = 4$)



(a) Core cells are shown in gray (b) Graph G (c) ϵ-neighbor cells (in gray) of the cell of o_{10}

Figure 3: DBSCAN with a grid ($MinPts = 4$)

- *(Connectivity) For any points $p_1, p_2 \in C$, there is a point $p \in C$ such that both p_1 and p_2 are density-reachable from p.*

Definition 3 implies that each cluster contains at least a core point (i.e., p). In Figure 2, $\{o_1, o_{10}\}$ is *not* a cluster because it does not involve all the points density-reachable from o_1. $\{o_1, o_2, o_3, ..., o_{10}\}$, on the other hand, is a cluster.

Ester et al. [10] gave a nice proof that P has a unique set of clusters, which gives rise to:

PROBLEM 1. *The* **DBSCAN problem** *is to find the unique set $\mathscr{C}$ of clusters of P.*

Given the input P in Figure 2, the problem should output two clusters: $C_1 = \{o_1, o_2, ..., o_{10}\}$ and $C_2 = \{o_{10}, o_{11}, ..., o_{17}\}$.

Remark. A cluster can contain both core and non-core points. Any non-core point p in a cluster is called a *border point*. Some points may not belong to any clusters at all; they are called *noise points*. In Figure 2, o_{10} is a border point, while o_{18} is noise.

The clusters in $\mathscr{C}$ are not necessarily disjoint (e.g., o_{10} belongs to both C_1 and C_2 in Figure 2). In general, if a point p appears in more than one cluster in $\mathscr{C}$, then p must be a border point (see Lemma 2 of [10]). In other words, a core point always belongs to a unique cluster.

2.2 The 2D Algorithm of [11]

Next, we explain in detail the algorithm of [11], which solves the DBSCAN problem in 2D space in $O(n \log n)$ time. The algorithm imposes an arbitrary grid T on the data space $\mathbb{R}^2$, where each cell of T is a $\frac{\epsilon}{\sqrt{2}} \times \frac{\epsilon}{\sqrt{2}}$ square. Figure 3a shows a grid on the data of Figure 2. Note that any two points in the same cell are at most distance ϵ apart. A cell c of T is *non-empty* if it contains at least one point of P; otherwise, c is *empty*. Clearly, there can be at most n non-empty cells.

The algorithm then launches a *labeling process* to decide for each point $p \in P$ whether p is core or non-core. Denote by $P(c)$ the set of points of P covered by c. A cell c is a *core cell* if $P(c)$ contains at least one core point. Denote by S_{core} the set of core cells in T. In Figure 3a where $MinPts = 4$, there are 6 core cells as shown in gray (core points are in black, and non-core points in white).

Let $G = (V, E)$ be a graph defined as follows:

- Each vertex in V corresponds to a distinct core cell in S_{core}.
- Given two different cells $c_1, c_2 \in S_{core}$, E contains an edge between c_1 and c_2 if and only if there exist *core* points $p_1 \in P(c_1)$ and $p_2 \in P(c_2)$ such that $dist(p_1, p_2) \leq \epsilon$.

Figure 3b shows the G for Figure 3a (note that there is no edge between cells c_4 and c_6).

The algorithm then proceeds by finding all the connected components of G. Let k be the number of connected components, V_i ($1 \leq i \leq k$) be the set of vertices in the i-th connected component, and $P(V_i)$ be the set of core points covered by the cells of V_i. Then:

LEMMA 1 ([11]). *The number k is also the number of clusters in P. Furthermore, $P(V_i)$ ($1 \leq i \leq k$) is exactly the set of core points in the i-th cluster.*

Figure 3b, $k = 2$, and $V_1 = \{c_1, c_2, c_3\}, V_2 = \{c_4, c_5, c_6\}$. It is easy to verify the correctness of Lemma 1 on this example.

Labeling Process. Let c_1 and c_2 be two different cells in T. They are *ϵ-neighbors* of each other if the minimum distance between them is at most ϵ. Figure 3c shows in gray all the ϵ-neighbor cells of the cell covering o_{10}. It is easy to see that each cell has at most 21 ϵ-neighbors. If a non-empty cell c contains at least $MinPts$ points, then *all* those points must be core points.

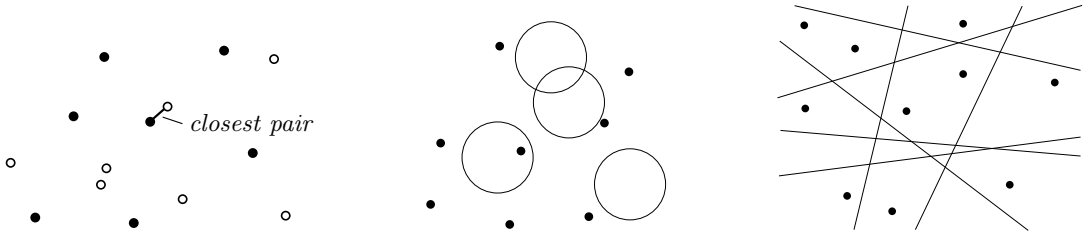
Now consider a cell c with $|P(c)| < MinPts$. Each point $p \in P(c)$ may or may not be a core point. To find out, the algorithm simply calculates the distances between p and *all* the points covered by *each* of the ϵ-neighbor cells of c. This allows us to know exactly the size of $|B(p, \epsilon)|$, and hence, whether p is core or non-core. For example, in Figure 3c, for $p = o_{10}$, we calculate the distance between o_{10} and all the points in the gray cells to find out that o_{10} is a non-core point.

Computation of G. Fix a core cell c_1. We will explain how to obtain the edges incident on c_1 in E. Let c_2 be a core cell that is an ϵ-neighbor of c_1. For each core point $p \in P(c_1)$, we find the core point $p' \in c_2$ that is the nearest to p. If $dist(p, p') \leq \epsilon$, an edge (c_1, c_2) is added to G. On the other hand, if all such $p \in P(c_1)$ have been tried but still no edge has been created, we conclude that E has no edge between c_1, c_2.

As a corollary of the above, each core cell c_1 has $O(1)$ incident edges in E (because it has $O(1)$ ϵ-neighbors). In other words, E has only a linear number $O(n)$ of edges.

Assigning Border Points. Recall that each $P(V_i)$ ($1 \leq i \leq k$) includes only the core points in the i-th cluster of P. It is still necessary to assign each non-core point q (i.e., border point) to the appropriate clusters. The principle of doing so is simple: *if p is a core point and $dist(p, q) \leq \epsilon$, then q should be added to the (unique) cluster of p.* To find all such core points p, Gunawan [11] adopted the following simple algorithm. Let c be the cell where q lies. For each ϵ-neighbor cell c' of c, simply calculate the distances from q to *all* the core points in c'.

Running Time. Gunawan [11] showed that, other than the computation of G, the rest of the algorithm runs in $O(MinPts \cdot$



(a) BCP (b) USEC (c) Hopcroft

Figure 4: Three relevant geometric problems

$n) = O(n)$ expected time or $O(n \log n)$ worst-case time. The computation of G requires $O(n)$ nearest neighbor queries, each of which can be answered in $O(\log n)$ time after building a Voronoi diagram for each core cell. Therefore, the overall execution time is bounded by $O(n \log n)$.

2.3 Some Geometric Results

Bichromatic Closest Pair (BCP). Let P_1, P_2 be two sets of points in $\mathbb{R}^d$ for some constant d. Set $m_1 = |P_1|$ and $m_2 = |P_2|$. The goal of the BCP problem is to find a pair of points $(p_1, p_2) \in P_1 \times P_2$ with the smallest distance, namely, $dist(p_1, p_2) \leq dist(p'_1, p'_2)$ for any $(p'_1, p'_2) \in P_1 \times P_2$. Figure 4 shows the closest pair for a set of black points and a set of white points.

In 2D space, it is well-known that BCP can be solved in $O(m_1 \log m_1 + m_2 \log m_2)$ time. The problem is much more challenging for $d \geq 3$, for which currently the best result is due to Agarwal et al. [1]:

LEMMA 2 ([1]). *For any fixed dimensionality $d \geq 4$, there is an algorithm solving the BCP problem in*

$$O\left((m_1 m_2)^{1-\frac{1}{\lceil d/2 \rceil + 1}+\delta'} + m_1 \log m_2 + m_2 \log m_1\right)$$

expected time, where $\delta' > 0$ can be an arbitrarily small constant. For $d = 3$, the expected running time can be improved to

$$O((m_1 m_2 \cdot \log m_1 \cdot \log m_2)^{2/3} + m_1 \log^2 m_2 + m_2 \log^2 m_1)).$$

Spherical Emptiness and Hopcroft. Let us now introduce the *unit-spherical emptiness checking* (USEC) *problem*:

> Let S_{pt} be a set of points, and S_{ball} be a set of balls *with the same radius*, all in data space $\mathbb{R}^d$, where the dimensionality d is a constant. The objective of USEC is to determine whether there is a point of S_{pt} that is covered by some ball in S_{ball}.

For example, in Figure 4b, the answer is *yes*.

Set $n = |S_{pt}| + |S_{ball}|$. In 3D space, the USEC problem can be solved in $O(n^{4/3} \cdot \log^{4/3} n)$ expected time [1]. Finding a 3D USEC algorithm with running time $o(n^{4/3})$ is a big open problem in computational geometry, and is widely believed to be impossible; see [8].

Strong hardness results are known about USEC when the dimensionality d is higher, owing to an established connection between the problem to the *Hopcroft's problem*:

> Let S_{pt} be a set of points, and S_{line} be a set of lines, all in data space $\mathbb{R}^2$ (note that the dimensionality is always 2). The goal of the Hopcroft's problem is to determine whether there is a point in S_{pt} that lies on some line of S_{line}.

For example, in Figure 4c, the answer is *no*.

The Hopcroft's problem can be settled in time slightly higher than $O(n^{4/3})$ time (see [18] for the precise bound), where $n = |S_{pt}| + |S_{line}|$. It is widely believed [8] that $\Omega(n^{4/3})$ is a lower bound on how fast the problem can be solved. In fact, this lower bound has already been proved on a broad class of algorithms [9].

It turns out that the Hopcroft's problem is a key reason of difficulty for a large number of other problems. This phenomenon gave rise to the notion of *Hopcroft hard* [8]. Specifically, a problem X is *Hopcroft hard* if an algorithm solving X in $o(n^{4/3})$ time implies an algorithm solving the Hopcroft's problem in $o(n^{4/3})$ time. In other words, a lower bound $\Omega(n^{4/3})$ on the time of solving the Hopcroft's problem implies the same lower bound on X.

Erickson [9] proved the following relationship between USEC and the Hopcroft's problem:

LEMMA 3 ([9]). *The USEC problem in any dimensionality $d \geq 5$ is Hopcroft hard.*

3. DBSCAN IN $\geq$ 3 DIMENSIONS

This section paves the way towards approximate DBSCAN, which is the topic of the next section. In Section 3.1, we establish the computational intractability of DBSCAN in practice via a novel reduction from the USEC problem (see Section 2.3). For practitioners that *insist* on applying this clustering method with the utmost accuracy, in Section 3.2, we present a new exact DBSCAN algorithm that outperforms all the previous solutions in time complexity.

3.1 Hardness of DBSCAN

We will prove:

> THEOREM 1. *The following statements are true about the DBSCAN problem:*
>
> - *It is Hopcroft hard in any dimensionality $d \geq 5$. Namely, the problem requires $\Omega(n^{4/3})$ time to solve, unless the Hopcroft problem can be settled in $o(n^{4/3})$ time.*
> - *When $d = 3$ (and hence, $d = 4$), the problem requires $\Omega(n^{4/3})$ time to solve, unless the USEC problem can be settled in $o(n^{4/3})$ time.*

As mentioned in Section 2.3, it is widely believed that neither the Hopcroft problem nor the USEC problem can be solved in $o(n^{4/3})$ time—any such algorithm would be a celebrated breakthrough in theoretical computer science.

Proof of Theorem 1. We observe a subtle connection between USEC and DBSCAN:

LEMMA 4. *For any dimensionality d, if we can solve the DBSCAN problem in $T(n)$ time, then we can solve the USEC problem in $T(n) + O(n)$ time.*

PROOF. Recall that the USEC problem is defined by a set S_{pt} of points and a set S_{ball} of balls with equal radii, both in $\mathbb{R}^d$. Denote

by $\mathcal{A}$ a DBSCAN algorithm in $\mathbb{R}^d$ that runs in $T(m)$ time on m points. Next, we describe an algorithm that deploys $\mathcal{A}$ as a *black box* to solve the USEC problem in $T(n) + O(n)$ time, where $n = |S_{pt}| + |S_{ball}|$.

Our algorithm is simple:

1. Obtain P, which is the union of S_{pt} and the set of centers of the balls in S_{ball}.
2. Set ϵ to the identical radius of the balls in S_{ball}.
3. Run $\mathcal{A}$ to solve the DBSCAN problem on P with this ϵ and $MinPts = 1$.
4. If any point in S_{pt} and any center of S_{ball} belong to the same cluster, then return *yes* for the USEC problem (namely, a point in S_{pt} is covered by some ball in S_{ball}). Otherwise, return *no*.

It is fundamental to implement the above algorithm in $T(n) + O(n)$ time. Next, we prove its correctness.

Case 1: We return yes. We will show that in this case there is indeed a point of S_{pt} that is covered by some ball in S_{ball}.

Recall that a *yes* return means a point $p \in S_{pt}$ and the center q of some ball in S_{ball} have been placed in the same cluster, which we denote by C. By connectivity of Definition 3, there exists a point $z \in C$ such that both p and q are density-reachable from z.

By setting $MinPts = 1$, we ensure that *all* the points in P are core points. In general, if a *core point* p_1 is density-reachable from p_2 (which by definition must be a core point), then p_2 is also density-reachable from p_1 (as can be verified by Definition 2). This means that z is density-reachable from p, which—together with the fact that q is density-reachable from z—shows that q is density-reachable from p.

It thus follows by Definition 2 that there is a sequence of points $p_1, p_2, ..., p_t \in P$ such that (i) $p_1 = p, p_t = q$, and (ii) $dist(p_i, p_{i+1}) \leq \epsilon$ for each $i \in [1, t-1]$. Let k be the smallest $i \in [2, t]$ such that p_i is the center of a ball in S_{ball}. Note that k definitely exists because p_t is such a center. It thus follows that p_{k-1} is a point from S_{pt}, and that p_{k-1} is covered by the ball in S_{ball} centered at p_k.

Case 2: We return no. We will show that in this case no point of S_{pt} is covered by any ball in S_{ball}.

This is in fact very easy. Suppose on the contrary that a point $p \in S_{pt}$ is covered by a ball of S_{ball} centered at q. Thus, $dist(p, q) \leq \epsilon$, namely, q is density-reachable from p. Then, by maximality of Definition 3, q must be in the cluster of p (recall that all the points of P are core points). This contradicts the fact that we returned *no*. □

Theorem 1 immediately follows from Lemmas 3 and 4.

3.2 A New Exact Algorithm for $d \geq 3$

It is well-known that DBSCAN can be solved in $O(n^2)$ time (e.g., see [25]) in any constant dimensionality d. Next, we show that it is possible to *always* terminate in $o(n^2)$ time regardless of d. Our algorithm extends that of [11] with two ideas:

- Use a d-dimensional grid T with an appropriate side length for its cells.
- Compute the edges of the graph G with a BCP algorithm (as opposed to nearest neighbor search).

Next, we explain the details. T is now a grid on $\mathbb{R}^d$ where each cell of T is a d-dimensional hyper-square with side length $\epsilon/\sqrt{d}$. As before, this ensures that any two points in the same cell are within distance ϵ from each other.

The algorithm description in Section 2.2 carries over to any $d \geq 3$ almost *verbatim*. The only difference is the way we compute the edges of G. Given core cells c_1 and c_2 that are ϵ-neighbors of each other, we solve the BCP problem on the sets of core points in c_1 and c_2, respectively. Let (p_1, p_2) be the pair returned. We add an edge (c_1, c_2) to G *if and only if* $dist(p_1, p_2) \leq \epsilon$.

The adapted algorithm achieves the following efficiency guarantee, whose proof is non-trivial, and can be found in the appendix.

THEOREM 2. *For any fixed dimensionality $d \geq 4$, there is an algorithm solving the DBSCAN problem in $O(n^{2-\frac{2}{\lceil d/2 \rceil + 1} + \delta})$ expected time, where $\delta > 0$ can be an arbitrarily small constant. For $d = 3$, the running time can be improved to $O((n \log n)^{4/3})$ expected.*

It is worth pointing out that the running time of our 3D algorithm nearly matches the lower bound in Theorem 1.

4. ρ-APPROXIMATE DBSCAN

The hardness result in Theorem 1 implies that DBSCAN is feasible only in 2D space. Even for $d = 3$, the computation time becomes strongly polynomial to n such that it will take an intolerably long period of time to calculate the clusters precisely. This opens the door to studying approximate DBSCAN.

In Section 4.1, we introduce the concept of ρ-approximate DBSCAN designed to replace DBSCAN on large datasets. In Section 4.2, we establish a strong quality guarantee of this new form of clustering. In Sections 4.3 and 4.4, we propose an algorithm for solving the ρ-approximate DBSCAN problem in time *linear* to the dataset size.

4.1 Definitions

As before, let P be the input set of n points in $\mathbb{R}^d$ to be clustered. We still take parameters ϵ and $MinPts$, but in addition, also a third parameter ρ, which can be any arbitrarily small positive constant, and controls the degree of approximation.

Next, we re-visit the basic definitions of DBSCAN in Section 2, and modify some of them to their "ρ-approximate versions". First, the notion of *core/non-core point* remains the same as Definition 1. The concept of *density-reachability* in Definition 2 is also inherited directly, but we will also need:

DEFINITION 4. *A point $q \in P$ is* **ρ-approximate density-reachable** *from $p \in P$ if there is a sequence of points $p_1, p_2, ..., p_t \in P$ (for some integer $t \geq 2$) such that:*

- *$p_1 = p$ and $p_t = q$*
- *$p_1, p_2, ..., p_{t-1}$ are core points*
- *$p_{i+1} \in B(p_i, \epsilon(1+\rho))$ for each $i \in [1, t-1]$.*

Note the difference between the above and Definition 2: in the third bullet, the radius of the ball is increased to $\epsilon(1+\rho)$. To illustrate, consider a small input set P as shown in Figure 5. Set $MinPts = 4$. The inner and outer circles have radii ϵ and $\epsilon(1+\rho)$, respectively. Core and non-core points are in black and white, respectively. Point o_5 is ρ-approximate density-reachable from o_3 (via sequence: o_3, o_2, o_1, o_5). However, o_5 is *not* density-reachable from o_3.

DEFINITION 5. *A* **ρ-approximate cluster** *C is a non-empty subset of P such that:*

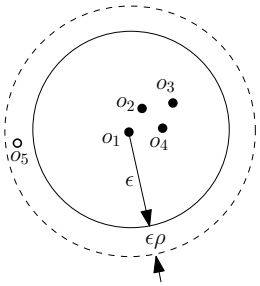


Figure 5: Density-reachability and ρ-approximate density-reachability ($MinPts = 4$)

- *(Maximality) If a core point $p \in C$, then all the points density-reachable from p also belong to C.*
- *(ρ-Approximate Connectivity) For any points $p_1, p_2 \in C$, there exists a point $p \in C$ such that both p_1 and p_2 are ρ-approximate density-reachable from p.*

Note the difference between the above and the original cluster formulation (Definition 1): the connectivity requirement has been weakened into ρ-approximate connectivity. In Figure 5, both $\{o_1, o_2, o_3, o_4\}$ and $\{o_1, o_2, o_3, o_4, o_5\}$ are ρ-approximate clusters.

PROBLEM 2. *The* **ρ-approximate DBSCAN problem** *is to find a set $\mathscr{C}$ of ρ-approximate clusters of P such that every core point of P appears in exactly one ρ-approximate cluster.*

Unlike the original DBSCAN problem, the ρ-approximate version may not have a unique result. In Figure 5, for example, it is legal to return either $\{o_1, o_2, o_3, o_4\}$ or $\{o_1, o_2, o_3, o_4, o_5\}$. Nevertheless, *any* result of the ρ-approximate problem comes with the quality guarantee to be proved next.

4.2 A Sandwich Theorem

Both DBSCAN and ρ-approximate DBSCAN are parameterized by ϵ and $MinPts$. It would be perfect if they can always return exactly the same clustering results. Of course, this is too good to be true. Nevertheless, in this subsection, we will show that this is *almost* true: the result of ρ-approximate DBSCAN is guaranteed to be somewhere between the (exact) DBSCAN results obtained by $(\epsilon, MinPts)$ and by $(\epsilon(1+\rho), MinPts)$! It is well-known that the clusters of DBSCAN rarely differ considerably when ϵ changes by just a small factor—in fact, if this really happens, it suggests that the choice of ϵ is very bad, such that the exact clusters are not stable anyway (we will come back to this issue later)!

Let us define:

- $\mathscr{C}_1$ as the set of clusters of DBSCAN with parameters $(\epsilon, MinPts)$
- $\mathscr{C}_2$ as the set of clusters of DBSCAN with parameters $(\epsilon(1+\rho), MinPts)$.
- $\mathscr{C}$ as an arbitrary set of clusters that is a legal result of $(\epsilon, MinPts, \rho)$-approx-DBSCAN.

The next theorem formalizes the quality assurance mentioned earlier:

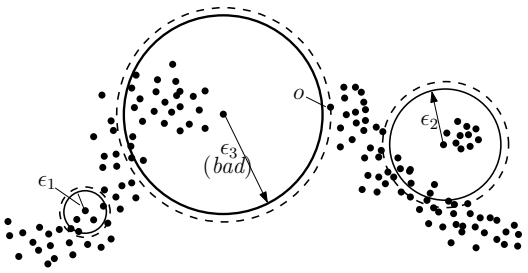


Figure 6: Good and bad choices of ϵ

THEOREM 3 (SANDWICH QUALITY GUARANTEE). *The following statements are true:*

1. *For any cluster $C_1 \in \mathscr{C}_1$, there is a cluster $C \in \mathscr{C}$ such that $C_1 \subseteq C$.*
2. *For any cluster $C \in \mathscr{C}$, there is a cluster $C_2 \in \mathscr{C}_2$ such that $C \subseteq C_2$.*

PROOF. To prove Statement 1, let p be an arbitrary core point in C_1. Then, C_1 is precisely the set of points in P density-reachable from p.[3] In general, if a point q is density-reachable from p in $(\epsilon, MinPts)$-exact-DBSCAN, q is also density-reachable from p in $(\epsilon, MinPts, \rho)$-approx-DBSCAN. By maximality of Definition 5, if C is the cluster in $\mathscr{C}$ containing p, then all the points of C_1 must be in C.

To prove Statement 2, consider an arbitrary core point $p \in C$ (there must be one by Definition 5). In $(\epsilon(1+\rho), MinPts)$-exact-DBSCAN, p must also be a core point. We choose C_2 to be the cluster of $\mathscr{C}_2$ where p belongs. Now, fix an arbitrary point $q \in C$. In $(\epsilon, MinPts, \rho)$-approx-DBSCAN, by ρ-approximate connectivity of Definition 5, we know that p and q are both ρ-approximate reachable from a point z. This implies that z is also ρ-approximate reachable from p. Hence, q is ρ-approximate reachable from p. This means that q is density-reachable from p in $(\epsilon(1+\rho), MinPts)$-exact-DBSCAN, indicating that $q \in C_2$. □

Here is an alternative, more intuitive, interpretation of Theorem 3:

- Statement 1 says that if two points belong to the same cluster of DBSCAN with parameters $(\epsilon, MinPts)$, they are *definitely* in the same cluster of ρ-approximate DBSCAN with the same parameters.
- On the other hand, a cluster of ρ-approximate DBSCAN parameterized by $(\epsilon, MinPts)$ may also contain two points p_1, p_2 that are in different clusters of DBSCAN with the same parameters. However, this is not bad because Statement 2 says that as soon as the parameter ϵ increases to $\epsilon(1+\rho)$, p_1 and p_2 will fall into the same cluster of DBSCAN!

Figure 6 nicely illustrates the effects of approximation. How many clusters are there? Interestingly, the answer is *it depends*. As pointed out in the classic OPTICS paper [2], different ϵ values allow us to view the dataset from various granularities, leading to different clustering results. In Figure 6, given ϵ_1 (and some

[3]This should be folklore but here is a proof. By maximality of Definition 3, all the points density-reachable from p are in C_1. On the other hand, let q be any point in C_1. By connectivity, p and q are both density-reachable from a point z. As p is a core point, we know that z is also density-reachable from p. Hence, q is density-reachable from p.

$MinPts$ say 2), DBSCAN outputs 3 clusters. Given ϵ_2, on the other hand, DBSCAN outputs 2 clusters, which makes sense because at this distance, the two clusters on the right merge into one.

Now let us consider approximation. The dashed circles illustrate the radii obtained with ρ-approximation. For both ϵ_1 and ϵ_2, ρ-approximate DBSCAN will return exactly the same clusters, because these distances are *robustly chosen* by being insensitive to small perturbation. For ϵ_3, however, ρ-approximate DBSCAN may return only one cluster (i.e., all points in the same cluster), whereas exact DBSCAN will return only two (i.e., the same two clusters as ϵ_2). By looking at the figure closely, one can realize that this happens because the dashed circle of radius $\rho(1+\epsilon_3)$ "happens" to pass a point—namely point o—which falls outside the solid circle of radius ϵ_3. Intuitively, ϵ_3 is a poor parameter choice because it is *too* close to the distance between two clusters such that a small change to it will cause the clustering results to be altered.

4.3 Approximate Range Counting

Let us now take a break from DBSCAN, and turn our attention to a different problem, whose solution is vital to our ρ-approximate DBSCAN algorithm.

Let P still be a set of n points in $\mathbb{R}^d$ where d is a constant. Given any point $q \in \mathbb{R}^d$, a distance threshold $\epsilon > 0$ and an arbitrarily small constant $\rho > 0$, an *approximate range count query* returns an integer that is guaranteed to be between $|B(q,\epsilon) \cap P|$ and $|B(q,\epsilon(1+\rho)) \cap P|$. For example, in Figure 5, given $q = o_1$, a query may return either 4 or 5.

Arya and Mount [3] developed a structure of $O(n)$ space that can be built in $O(n \log n)$ time, and answers any such query in $O(\log n)$ time. Next, we design an alternative structure with better performance in our context:

LEMMA 5. *For any fixed ϵ and ρ, there is a structure of $O(n)$ space that can be built in $O(n)$ expected time, and answers any approximate range count query in $O(1)$ expected time.*

Structure. Our structure is a simple quadtree-like hierarchical grid partitioning of $\mathbb{R}^d$. First, impose a regular grid on $\mathbb{R}^d$ where each cell is a d-dimensional hyper-square with side length $\epsilon/\sqrt{d}$. For each *non-empty* cell c of the grid (i.e., c covers at least 1 point of P), divide it into 2^d cells of the same size. For each resulting *non-empty* cell c', divide it recursively in the same manner, until the side length of c' is at most $\epsilon\rho/\sqrt{d}$.

We use H to refer to the hierarchy thus obtained. We keep *only* the non-empty cells of H, and for each such cell c, record $cnt(c)$ which is the number of points in P covered by c. We will refer to a cell of H with side length $\epsilon/(2^i\sqrt{d})$ as a *level-i cell*. Clearly, H has only $h = \max\{1, 1 + \lceil \log_2(1/\rho) \rceil\} = O(1)$ levels. If a level-$(i+1)$ cell c' is inside a level-i cell c, we say that c' is a *child* of c, and c a *parent* of c'. A cell with no children is called a *leaf cell*.

Figure 7 illustrates the part of the first three levels of H for the dataset on the left. Note that empty cells are *not* stored.

Query. Given an approximate range count query with parameters q, ϵ, ρ, we compute its answer ans as follows. Initially, $ans = 0$. In general, given a non-empty level-i cell c, we distinguish three cases:

- If c is disjoint with $B(q,\epsilon)$, ignore it.
- If c is fully covered by $B(q,\epsilon(1+\rho))$, add $cnt(c)$ to ans.
- When neither of the above holds, check if c is a leaf cell in H. If not, process the child cells of c in the same manner. Otherwise (i.e., c is a leaf), add $cnt(c)$ to ans only if c intersects $B(q,\epsilon)$.

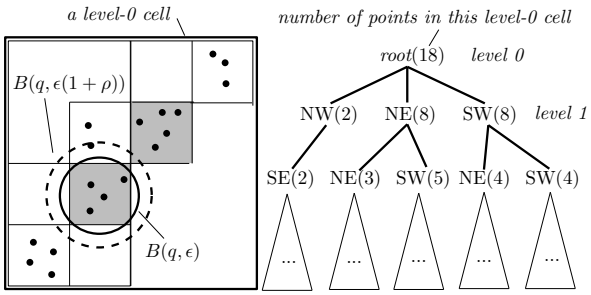


Figure 7: Approximate range counting

The algorithm starts from the level-0 non-empty cells that intersect with $B(q,\epsilon)$.

To illustrate, consider the query shown in Figure 7. The two gray cells correspond to nodes SW(5) and NE(4) at level 2. The subtree of *neither* of them is visited, but the reasons are different. For SW(5), its cell is disjoint with $B(q,\epsilon)$, so we ignore it (even though it intersects $B(q,\epsilon(1+\rho))$). For NE(4), its cell completely falls in $B(q,\epsilon(1+\rho))$, so we add its count 4 to the result (even though it is not covered by $B(q,\epsilon)$).

Correctness. The above algorithm has two guarantees. First, if a point $p \in P$ is inside $B(q,\epsilon)$, it is definitely counted in ans. Second, if p is outside $B(q,\epsilon(1+\rho))$, then it is definitely *not* counted in ans. These guarantees are easy to verify, utilizing the fact that if a leaf cell c intersects $B(p,\epsilon)$, then c must fall *completely* in $B(p,\epsilon(1+\rho))$ because any two points in a leaf cell are within distance $\epsilon\rho$. It thus follows that the ans returned is a legal answer.

Time Analysis. Remember that the hierarchy H has $O(1)$ levels. Since there are $O(n)$ non-empty cells at each level, the total space is $O(n)$. With hashing, it is easy to build the structure level by level in $O(n)$ expected time.

To analyze the running time of our query algorithm, observe that each cell c visited by our algorithm must satisfy one of the following conditions: (i) c is a level-0 cell, or (ii) the parent of c intersects the boundary of $B(q,\epsilon)$. For type-(i), the $O(1)$ level-0 cells intersecting $B(q,\epsilon)$ can be found in $O(1)$ expected time using the coordinates of q. For type-(ii), it suffices to bound the number of cells intersecting the boundary of $B(q,\epsilon)$ because each such cell has $2^d = O(1)$ child nodes.

In general, a d-dimensional grid of cells with side length l has $O(1 + (\frac{\theta}{l})^{d-1})$ cells intersecting the boundary of a sphere with radius θ [3]. Combining this and the fact that a level-i cell has side length $\epsilon/(2^i\sqrt{d})$, we know that the total number of cells (of all levels) intersecting the boundary of $B(q,\epsilon)$ is bounded by:

$$\sum_{i=0}^{h-1} O\left(1 + \left(\frac{\epsilon}{\epsilon/(2^i\sqrt{d})}\right)^{d-1}\right) = O\left((2^h)^{d-1}\right)$$
$$= O\left(1 + (1/\rho)^{d-1}\right)$$

which is a constant for any fixed ρ. This concludes the proof of Lemma 5.

4.4 Solving ρ-Approximate DBSCAN

We are now ready to solve the ρ-approximate DBSCAN problem by proving our last main result:

THEOREM 4. *There is a ρ-approximate DBSCAN algorithm that terminates in $O(n)$ expected time, regardless of the value of ϵ, the constant approximation ratio ρ, and the fixed dimensionality d.*

Algorithm. Our ρ-approximate algorithm differs from the exact algorithm we proposed in Section 3.2 *only* in the definition and computation of the graph G. We re-define $G = (V, E)$ as follows:

- As before, each vertex in V is a core cell of the grid T (remember that the algorithm of Section 3.2 imposes a grid T on $\mathbb{R}^d$, where a cell is a core cell if it covers at least one core point).
- Given two different core cells c_1, c_2, whether E has an edge between c_1 and c_2 obeys the rules below:
  - yes, if there exist core points p_1, p_2 in c_1, c_2, respectively, such that $dist(p_1, p_2) \leq \epsilon$.
  - no, if no core point in c_1 is within distance $\epsilon(1 + \rho)$ from any core point in c_2.
  - *don't care*, in all the other cases.

To compute G, our algorithm starts by building, for each core cell c in T, a structure of Lemma 5 on the set of core points in c. To generate the edges of a core cell c_1, we examine each ϵ-neighbor cell c_2 of c_1 in turn. For every core point p in c_1, do an approximate range count query on the set of core points in c_2. If the query returns a non-zero answer, add an edge (c_1, c_2) to G. If all such p have been tried but still no edge has been added, we decide that there should be no edge between c_1 and c_2.

Correctness. Let C be an arbitrary cluster returned by our algorithm. We will show that C satisfies Definition 5.

Maximality. Let p be an arbitrary core point in C, and q be any point of P density-reachable from p. We will show that $q \in C$. Let us start by considering that q is a core point. By Definition 2, there is a sequence of core points $p_1, p_2, ..., p_t$ (for some integer $t \geq 2$) such that $p_1 = p$, $p_t = q$, and $dist(p_{i+1}, p_i) \leq \epsilon$ for each $i \in [1, t-1]$. Denote by c_i the cell of T covering p_i. By the way G is defined, there must be an edge between c_i and c_{i+1}, for each $i \in [1, t-1]$. It thus follows that c_1 and c_t must be in the same connected component of G; therefore, p and q must be in the same cluster. The correctness of the other scenario where q is a non-core point is trivially guaranteed by the way that non-core points are assigned to clusters.

ρ-Approximate Connectivity. Let p be an arbitrary core point in C. For *any* point $q \in C$, we will show that q is ρ-approximate density-reachable from p. Again, we consider first that q is a core point. Let c_p and c_q be the cells of T covering p and q, respectively. Since c_p and c_q are in the same connected component of G, there is a path $c_1, c_2, ..., c_t$ in G (for some integer $t \geq 2$) such that $c_1 = c_p$ and $c_t = c_q$. Recall that any two points in the same cell are within distance ϵ. Combining this fact with how the edges of G are defined, we know that there is a sequence of core points $p_1, p_2, ..., p_{t'}$ (for some integer $t' \geq 2$) such that $p_1 = p$, $p_{t'} = q$, and $dist(p_{i+1}, p_i) \leq \epsilon(1+\rho)$ for each $i \in [1, t'-1]$. Therefore, q is ρ-approximate density-reachable from p. The correctness of the other scenario where q is a non-core point is again trivial.

Time Analysis. It takes $O(n)$ expected time to construct the structure of Lemma 5 for all cells. The expected time of computing G is proportional to the number of approximate range count queries issued. For each core point of a cell c_1, we issue $O(1)$ queries in total (one for each ϵ-neighbor cell of c_2). Hence, the total number of queries is $O(n)$. The rest of the ρ-approximate algorithm runs in $O(n)$ expected time, following the same analysis in [11]. This completes the proof of Theorem 4. It is worth mentioning that, intuitively, the efficiency improvement of our approximate algorithm (over the exact algorithm in Section 3.2) owes to the fact that we settle for an imprecise solution to the BCP problem by using Lemma 5.

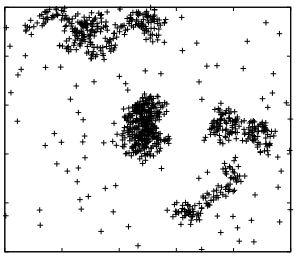


Figure 8: A 2D seed spreader dataset

5. EXPERIMENTS

We now present an empirical evaluation of the proposed techniques. All the experiments were run on a machine equipped with 3.2GHz CPU and 8 GB memory. The operating system was Linux (Ubuntu 13.04). All the programs were coded in C++, and compiled using g++ with -o3 turned on.

Section 5.1 describes the datasets in our experimentation, after which we present our findings in two parts. First, Section 5.2 assesses the clustering precision of ρ-approximate DBSCAN. Then, Section 5.3 demonstrates the vast efficiency gain achieved by our approximation algorithm compared to exact DBSCAN.

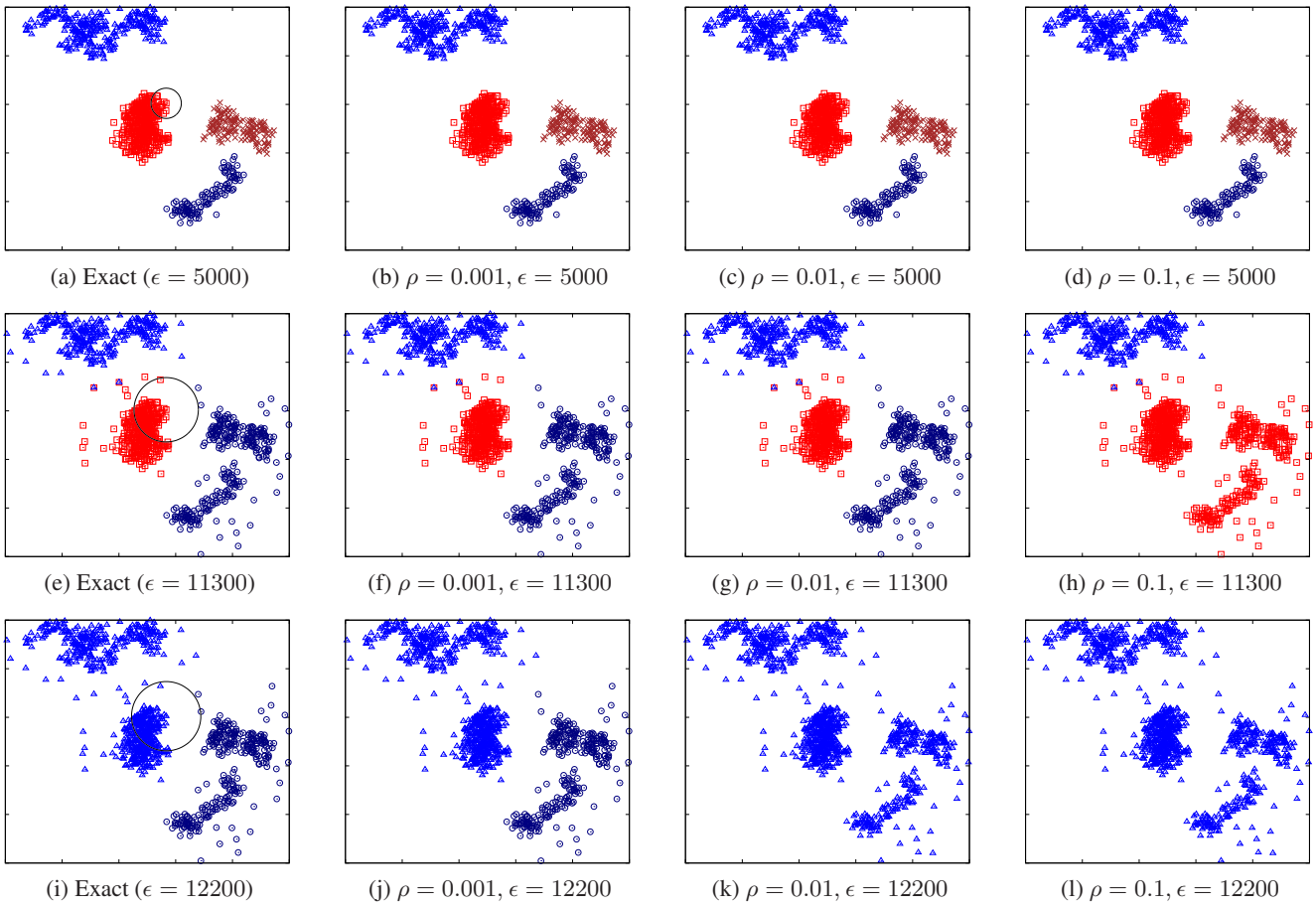
5.1 Datasets

Except in a single experiment (for visualization), we focused on dimensionality $d \geq 3$ because the 2D case has been well solved in [11]. In all cases, the underlying data space had a normalized domain of $[0, 10^5]$ for every dimension. We deployed both synthetic and real datasets whose details are explained next.

Synthetic: Seed Spreader (SS). A synthetic dataset was generated in a "random walk with restart" fashion. First, fix the dimensionality d, take the target cardinality n, a *restart probability* $\rho_{restart}$, and a *noise percentage* ρ_{noise}. Then, we simulate a *seed spreader* that moves about in the space, and spits out data points around its current location. The spreader carries a *local counter* such that whenever the counter reaches 0, the spreader moves a distance of r_{shift} towards a random direction, after which the counter is reset to c_{reset}. The spreader works in *steps*. In each step, (i) with probability $\rho_{restart}$, the spreader *restarts*, by jumping to a random location in the data space, and resetting its counter to c_{reset}; (ii) no matter if a restart has happened, the spreader produces a point uniformly at random in the ball centered at its current location with radius 100, after which the local counter decreases by 1. Intuitively, every time a restart happens, the spreader begins to generate a new cluster. In the first step, a restart is forced so as to put the spreader at a random location. We repeat in total $n(1 - \rho_{noise})$ steps, which generate the same number of points.

parameter	values
n (synthetic)	100k, 0.5m, 1m, **2m**, 5m, 10m
d (synthetic)	3, **5**, 7
ϵ	from **5000** to the collapsing radius
ρ	from **0.001**, 0.01, 0.02, ..., 0.1

Table 1: Parameter values (defaults in bolds)



(a) Exact ($\epsilon = 5000$) (b) $\rho = 0.001, \epsilon = 5000$ (c) $\rho = 0.01, \epsilon = 5000$ (d) $\rho = 0.1, \epsilon = 5000$

(e) Exact ($\epsilon = 11300$) (f) $\rho = 0.001, \epsilon = 11300$ (g) $\rho = 0.01, \epsilon = 11300$ (h) $\rho = 0.1, \epsilon = 11300$

(i) Exact ($\epsilon = 12200$) (j) $\rho = 0.001, \epsilon = 12200$ (k) $\rho = 0.01, \epsilon = 12200$ (l) $\rho = 0.1, \epsilon = 12200$

Figure 9: Comparison of the clusters found by exact DBSCAN and ρ-approximate DBSCAN

Finally, we add $n \cdot \rho_{noise}$ noise points, each of which is uniformly distributed in the whole space.

Figure 8 shows a small 2D dataset which was generated with $n = 1000$ and had 4 restarts; the dataset will be used for visualization. In all the other experiments, $c_{reset} = 100$, $r_{shift} = 50d$, $\rho_{restart}$ was fixed to $10/(n(1 - \rho_{noise}))$ with $\rho_{noise} = 1/10^4$. In expectation, around 10 restarts would occur in the generation. The value of n ranged from 100k all the way to 10 million, while d from 3 to 7. See Table 1.

Real. Three real datasets were employed in our experimentation. The first one, *PAMAP2*, is a 4-dimensional dataset with cardinality 3,850,505, obtained by taking the first 4 principle components of a PCA on the PAMAP2 database [22] from the UCI machine learning archive [4]. The second one, *Farm*, is a 5-dimensional dataset with cardinality 3,627,086, which contains the VZ-features [27] of a satellite image of a farm in Saudi Arabia[4]. It is worth noting that VZ-feature clustering is a common approach to perform color segmentation of an image [27]. The third one, *Household*, is a 7-dimensional dataset with cardinality 2,049,280, which includes all the attributes of the Household database again from the UCI archive [4] except the temporal columns *date* and *time*. Points in the original database with missing coordinates were removed.

Collapsing Radius. The parameter $MinPts$ was fixed to 100 in all cases (except only the visualization experiment). Every dataset has a unique *collapsing radius*, which is the smallest ϵ such that exact DBSCAN returns a single cluster. For each dataset, we inspected a wide range of ϵ from 5000 all the way to its collapsing radius.

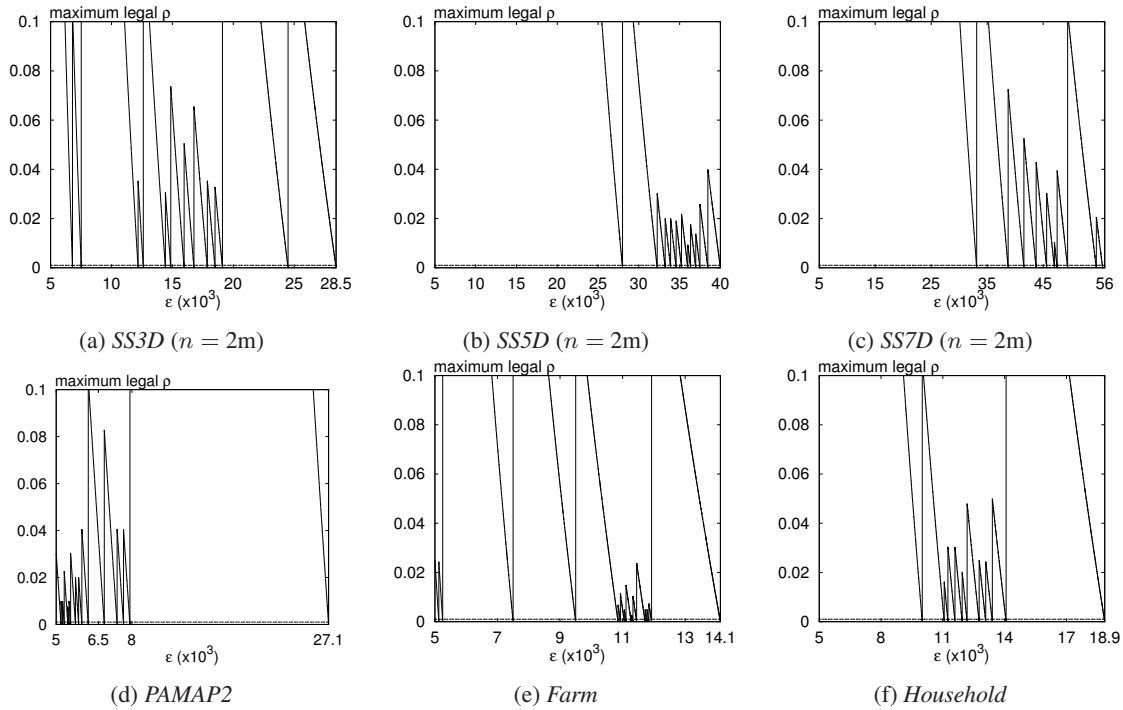
[4] *http://www.satimagingcorp.com/gallery/ikonos/ikonos-tadco-farms-saudi-arabia*

5.2 Approximation Quality

2D Visualization. Let us start by showing to the reader directly the effects of approximation. For this purpose, we take the 2D dataset in Figure 8 as the input (note that the cardinality was deliberately chosen to be small to facilitate visualization), and fixed $MinPts = 20$. Figure 9a demonstrates the 4 clusters found by exact DBSCAN with $\epsilon = 5000$ (which is the radius of the circle shown). The points of each cluster are depicted with the same color and marker. Figures 9b, 9c, and 9d present the clusters found by our ρ-approximate DBSCAN when ρ equals 0.001, 0.01, and 0.1, respectively. In all cases, ρ-approximate DBSCAN returned *exactly the same* clusters as DBSCAN.

Making things more interesting, in Figure 9e, we increased ϵ to 11300 (again, ϵ is the radius of the circle shown). This time, DBSCAN found 3 clusters (note that 2 clusters in Figure 9a have merged). Figures 9f, 9g, and 9h give the clusters of ρ-approximate DBSCAN for $\rho = 0.001$, 0.01, and 0.1, respectively. Once again, the clusters of $\rho = 0.001$ and 0.01 are exactly the same as DBSCAN. However, 0.1-approximate DBSCAN returned only 2 clusters. This can be understood by observing that the circle in Figure 9e almost touched a point from a different cluster. In fact, it will, once ϵ increases by 10%, which explains why 0.1-approximate DBSCAN produced different results.

Then we pushed ϵ even further to 12200 so that DBSCAN yielded 2 clusters as shown in Figure 9i. Figures 9j, 9k, and 9l



(a) *SS3D* ($n = 2$m) (b) *SS5D* ($n = 2$m) (c) *SS7D* ($n = 2$m)

(d) *PAMAP2* (e) *Farm* (f) *Household*

Figure 10: Maximum legal ρ vs. radius ϵ together with the line $\rho = 0.001$

illustrate the clusters of ρ-approximate DBSCAN for $\rho = 0.001$, 0.01, and 0.1, respectively. Here, both $\rho = 0.01$ and 0.1 had given up, but $\rho = 0.001$ still robustly churned out exactly the same clusters as DBSCAN.

Surprised by $\rho = 0.01$ not working, we examined carefully the reason behind its failure, and then, realized something interesting. It turned out that 12200 was *extremely* close to the "boundary ϵ" for DBSCAN to output 2 clusters. Specifically, as soon as ϵ grew up to 12203, the exact DBSCAN would return only a single cluster! Actually, this could be seen from Figure 9i—note how close the circle is to the point from the right cluster! In other words, 12200 is in fact an "unstable" value for ϵ.

All Dimensionalities—A Sawtooth View. How to evaluate the approximation quality as we depart from 2D? What is described next essentially puts ρ-approximate DBSCAN under strict scrutiny. Fix a dataset. For each ϵ, we define the *maximum legal* ρ at ϵ as the largest ρ under which ρ-approximate DBSCAN returns *exactly the same* clusters as DBSCAN. Figures 10a, 10b, and 10c plot the maximum legal ρ as a function of ϵ, for the synthetic SS (seed spreader) datasets of $d = 3, 5, 7$ under the default n = 2m, respectively. In each diagram, the range of ϵ covers the entire spectrum as indicated in Table 1. Figures 10d, 10e, and 10f present the results of the same experiment on the three real datasets, respectively.

In general, the maximum legal ρ exhibits a sawtooth shape as ϵ increases. For most ϵ values, the maximum legal ρ is much higher than 0.1 (this is so whenever the curve disappears from the top of a diagram). We recommend setting $\rho = 0.001$, which is also the default in all our experiments. As is evident in Figure 10 (where the horizontal line near the bottom of each diagram represents $\rho = 0.001$), 0.001 is below the maximum legal ρ almost for all values of ϵ. In other words, $\rho = 0.001$ guarantees the same results as exact DBSCAN almost everywhere. In fact, whenever 0.001 is higher than the maximum legal ρ, the value of ϵ falls in a tiny range in which exact DBSCAN produces different clusters. In other words, those values of ϵ are unstable anyway. This can also be seen from sandwich theorem (Theorem 3), namely, the results of 0.001-approximate DBSCAN must fall between the results of DBSCAN with ϵ and 1.001ϵ, respectively. Hence, if 0.001-approximate DBSCAN differs from DBSCAN, it means that the results of DBSCAN have changed within $[\epsilon, 1.001\epsilon]$!

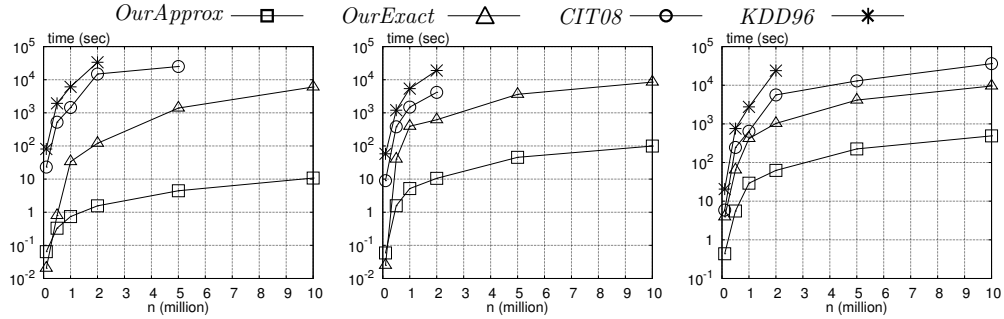
5.3 Computational Efficiency

We now proceed to inspect the running time of DBSCAN clustering using four algorithms:

- *KDD96* [10]: the original DBSCAN algorithm in [10];
- *CIT08* [17]: the state of the art of exact DBSCAN, namely, the fastest existing algorithm able to produce the same DBSCAN result as *KDD96*;
- *OurExact*: the exact DBSCAN algorithm we developed in Theorem 2;
- *OurApprox*: the ρ-approximate DBSCAN algorithm we proposed in Theorem 4.

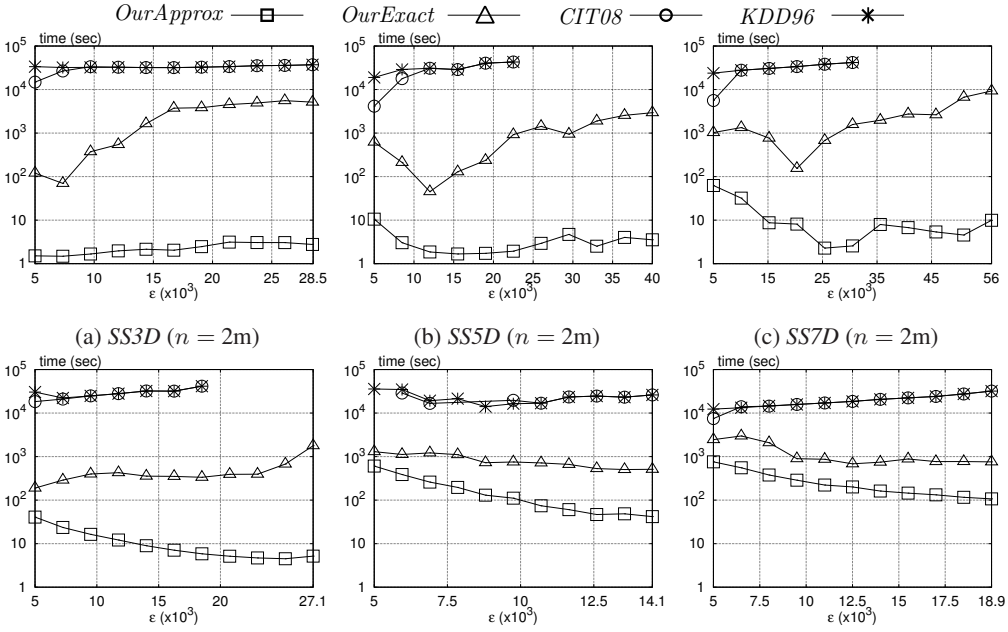
Scalability with Cardinality n. The first experiment examines how each method scales with the number n objects. For this purpose, we used synthetic SS datasets of 3D, 5D, and 7D by varying n from 100k to 10m. The results are presented in Figure 11—note that the y-axis is in log scale. If *KDD96* and *CIT08* do not have results at a value of n, it means that they did not terminate within 12 hours in the corresponding experiments! *OurExact* managed to finish within 10^4 seconds (less than 3 hours) even on the largest dataset. However, this is dwarfed by the superb efficiency of *OurApprox* which took less than 490 seconds in all cases, and were often faster than *OurExact* by a factor of two orders of magnitude.

As an interesting note, all methods were fast when the dataset was small, e.g., when $n = 100$k. This is perhaps the reason why most (if not all) of the previous evaluation of DBSCAN algorithms, as far as we are aware, was on datasets of this scale.



(a) *SS3D* (b) *SS5D* (c) *SS7D*

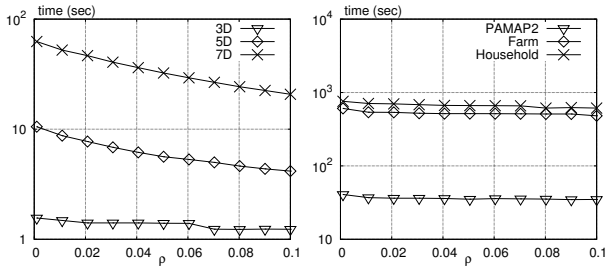
Figure 11: Running time vs. cardinality n ($\epsilon = 5000$, $\rho = 0.001$)



(a) *SS3D* ($n = 2$m) (b) *SS5D* ($n = 2$m) (c) *SS7D* ($n = 2$m)

(d) *PAMAP2* (e) *Farm* (f) *Household*

Figure 12: Running time vs. radius ϵ ($\rho = 0.001$)



(a) SS datasets ($n = 2$m) (b) Real datasets

Figure 13: Running time vs. approximation ratio ρ ($\epsilon = 5000$)

Influence of Radius ϵ. The next experiment aimed to understand the behavior of each method under the influence of ϵ. Figure 12 plots the running time as a function of ϵ, when this parameter varied in the corresponding spectrum of various datasets. If *KDD96* and *CIT08* did not terminate within 12 hours, they have no results. In general, the cost of both *KDD96* and *CIT08* increases with ϵ because both methods rely on *range queries*, each retrieving all the points in a circle, and thus, becoming more expensive as the circle grows. However, *OurExact* and *OurApprox* do not have such monotonic behavior. The effects of ϵ on the two methods are *heavily* dependent on the data distribution. In any case, *OurApprox* consistently outperformed all other methods significantly in Figure 12.

Influence of Approximation Ratio ρ. Finally, Figure 13 shows the running time of *OurApprox* as a function of ρ, on the 3D, 5D, 7D SS datasets of size 2m and the real datasets. Very much as expected, when ρ increased (i.e., less precision is demanded), *OurApprox* became more efficient.

6. CONCLUSIONS

DBSCAN is a creative, elegant, and effective technique for density-based clustering, which is very extensively applied in data mining, machine learning, and databases. However, all the existing DBSCAN algorithms have poor scalability with the dataset size in dimensionality $d \geq 3$. This is unfortunate because clustering in $d \geq 3$ is important, due to the frequent need of using multiple features to accurately model an object.

In this paper, we explain rigorously why the community's search for a fast DBSCAN algorithm for $d \geq 3$ has been unsuccessful. We show that, unless very significant breakthroughs (ones widely believed to be impossible) can be made in theoretical computer science, the DBSCAN algorithm requires $\Omega(n^{4/3})$ time

to solve, where n is the size of the underlying dataset. This strongly polynomial complexity essentially states that DBSCAN is computationally intractable in practice, even for moderately large n! This is very disappointing especially given the arrival of the big data era. Motivated by this, we propose the novel concept of ρ-approximate DBSCAN, which is designed to replace DBSCAN on large-scale data. We prove both theoretical and experimentally that ρ-approximate DBSCAN has excellent guarantees both in the quality of cluster approximation and computational efficiency. In fact, almost in all scenarios, it returns exactly the same clusters as DBSCAN but requires computation time only linear to n.

7. REFERENCES

[1] P. K. Agarwal, H. Edelsbrunner, and O. Schwarzkopf. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete & Computational Geometry*, 6:407–422, 1991.

[2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD*, pages 49–60, 1999.

[3] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry*, 17(3-4):135–152, 2000.

[4] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[5] C. Böhm, K. Kailing, P. Kröger, and A. Zimek. Computing clusters of correlation connected objects. In *SIGMOD*, pages 455–466, 2004.

[6] B. Borah and D. K. Bhattacharyya. An improved sampling-based DBSCAN for large spatial databases. In *Proceedings of Intelligent Sensing and Information Processing*, pages 92–96, 2004.

[7] V. Chaoji, M. A. Hasan, S. Salem, and M. J. Zaki. Sparcl: Efficient and effective shape-based clustering. In *ICDM*, pages 93–102, 2008.

[8] J. Erickson. On the relative complexities of some geometric problems. In *CCCG*, pages 85–90, 1995.

[9] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete & Computational Geometry*, 16(4):389–418, 1996.

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[11] A. Gunawan. A faster algorithm for DBSCAN. Master's thesis, Technische University Eindhoven, March 2013.

[12] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2012.

[13] M. Klusch, S. Lodi, and G. Moro. Distributed clustering based on sampling local density estimates. In *IJCAI*, pages 485–490, 2003.

[14] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.

[15] B. Liu. A fast density-based clustering algorithm for large databases. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pages 996–1000, 2006.

[16] E. H.-C. Lu, V. S. Tseng, and P. S. Yu. Mining cluster-based temporal mobile sequential patterns in location-based service environments. *TKDE*, 23(6):914–927, 2011.

[17] S. Mahran and K. Mahar. Using grid for accelerating density-based clustering. In *CIT*, pages 35–40, 2008.

[18] J. Matousek. Range searching with efficient hiearchical cutting. *Discrete & Computational Geometry*, 10:157–182, 1993.

[19] B. L. Milenova and M. M. Campos. O-Cluster: Scalable clustering of large high dimensional data sets. In *ICDM*, pages 290–297, 2002.

[20] S. K. Pal and P. Mitra. *Pattern Recognition Algorithms for Data Mining*. Chapman and Hall/CRC, 2004.

[21] T. Pei, A.-X. Zhu, C. Zhou, B. Li, and C. Qin. A new approach to the nearest-neighbour method to discover cluster features in overlaid spatial point processes. *International Journal of Geographical Information Science*, 20(2):153–168, 2006.

[22] A. Reiss and D. Stricker. Introducing a new benchmarked dataset for activity monitoring. In *International Symposium on Wearable Computers*, pages 108–109, 2012.

[23] S. Roy and D. K. Bhattacharyya. An approach to find embedded clusters using density based techniques. In *Proceedings of Distributed Computing and Internet Technology*, pages 523–535, 2005.

[24] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A wavelet based clustering approach for spatial data in very large databases. *VLDB J.*, 8(3-4):289–304, 2000.

[25] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson, 2006.

[26] C.-F. Tsai and C.-T. Wu. GF-DBSCAN: A new efficient and effective data clustering technique for large databases. In *Proceedings of International Conference on Multimedia Systems and Signal Processing*, pages 231–236, 2009.

[27] M. Varma and A. Zisserman. Texture classification: Are filter banks necessary? In *CVPR*, pages 691–698, 2003.

[28] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In *VLDB*, pages 186–195, 1997.

[29] J.-R. Wen, J.-Y. Nie, and H. Zhang. Query clustering using user logs. *TOIS*, 20(1):59–81, 2002.

[30] C. Zhou, D. Frankowski, P. J. Ludford, S. Shekhar, and L. G. Terveen. Discovering personally meaningful places: An interactive clustering approach. *TOIS*, 25(3), 2007.

Proof of Theorem 2

It suffices to analyze the time used by our algorithm to generate the edges of G. The other parts of the algorithm use $O(n)$ expected time, following the analysis of [11].

Let us consider first $d \geq 4$. First, fix the value of δ in Theorem 2. Define: $\lambda = \frac{1}{\lceil d/2 \rceil + 1} - \delta/2$. Given a core cell c, we denote by m_c the number of core points in c. Then, by Lemma 2, the time we spend generating the edges of G is

$$\sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c'}} O\left((m_c m_{c'})^{1-\lambda} + m_c \log m_{c'} + m_{c'} \log m_c \right). \quad (1)$$

To bound the first term, we derive

$$\begin{aligned}
&\sum_{\epsilon\text{-neighbor core cells } c,\, c'} O\left((m_c m_{c'})^{1-\lambda} \right) \\
&= \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c \leq m_{c'}}} O\left((m_c m_{c'})^{1-\lambda} \right) + \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c > m_{c'}}} O\left((m_c m_{c'})^{1-\lambda} \right) \\
&= \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c \leq m_{c'}}} O\left(m_{c'} \cdot m_c^{1-2\lambda} \right) + \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c > m_{c'}}} O\left(m_c \cdot m_{c'}^{1-2\lambda} \right) \\
&= \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c \leq m_{c'}}} O\left(m_{c'} \cdot n^{1-2\lambda} \right) + \sum_{\substack{\epsilon\text{-neighbor} \\ \text{core cells } c,\, c' \\ \text{s.t. } m_c > m_{c'}}} O\left(m_c \cdot n^{1-2\lambda} \right) \\
&= O\Big(n^{1-2\lambda} \sum_{\epsilon\text{-neighbor core cells } c,\, c'} m_c \Big) = O(n^{2-2\lambda})
\end{aligned}$$

where the last equality used the fact that c has only $O(1)$ ϵ-neighbor cells as long as d is a constant (and hence, m_c can be added only $O(1)$ times). The other terms in (1) are easy to bound:

$$\begin{aligned}
&\sum_{\epsilon\text{-neighbor core cells } c,\, c'} O\left(m_c \log m_{c'} + m_{c'} \log m_c \right) \\
&= \sum_{\epsilon\text{-neighbor core cells } c,\, c'} O\left(m_c \log n + m_{c'} \log n \right) = O(n \log n).
\end{aligned}$$

In summary, we spend $O(n^{2-2\lambda} + n \log n) = O(n^{2 - \frac{2}{\lceil d/2 \rceil + 1} + \delta})$ time generating the edges of E. This proves the part of Theorem 2 for $d \geq 4$. An analogous analysis based on the $d = 3$ branch of Lemma 2 establishes the other part of Theorem 2.