DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN

ERICH SCHUBERT, Heidelberg University
JÖRG SANDER, University of Alberta
MARTIN ESTER, Simon Fraser University
HANS-PETER KRIEGEL, Ludwig-Maximilians-Universität München
XIAOWEI XU, University of Arkansas at Little Rock

At SIGMOD 2015, an article was presented with the title "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation" that won the conference's best paper award. In this technical correspondence, we want to point out some inaccuracies in the way DBSCAN was represented, and why the criticism should have been directed at the assumption about the performance of spatial index structures such as R-trees and not at an algorithm that can use such indexes. We will also discuss the relationship of DBSCAN performance and the indexability of the dataset, and discuss some heuristics for choosing appropriate DBSCAN parameters. Some indicators of bad parameters will be proposed to help guide future users of this algorithm in choosing parameters such as to obtain both meaningful results and good performance. In new experiments, we show that the new SIGMOD 2015 methods do not appear to offer practical benefits if the DBSCAN parameters are well chosen and thus they are primarily of theoretical interest. In conclusion, the original DBSCAN algorithm with effective indexes and reasonably chosen parameter values performs competitively compared to the method proposed by Gan and Tao.

CCS Concepts: • **Computing methodologies** → **Cluster analysis**; • **Information systems** → **Clustering**; • **Theory of computation** → *Unsupervised learning and clustering*;

Additional Key Words and Phrases: DBSCAN, density-based clustering, range-search complexity

ACM Reference format:
Erich Schubert, Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst.* 42, 3, Article 19 (July 2017), 21 pages.
https://doi.org/10.1145/3068335

Authors' addresses: E. Schubert, Heidelberg University, Institut für Informatik, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany; email: schubert@informatik.uni-heidelberg.de; J. Sander, University of Alberta, Department of Computing Science, Athabasca Hall 2-21, Edmonton, AB, Canada T6G 2E8; email: jsander@ualberta.ca; M. Ester, Simon Fraser University, School of Computing Science, 8888 University Drive, Burnaby, B.C. Canada V5A 1S6; email: ester@cs.sfu.ca; H.-P. Kriegel, Ludwig-Maximilians-Universität München, Oettingenstraße 67, 80538 München, Germany; email: kriegel@dbs.ifi.lmu.de; X. Xu, University of Arkansas at Little Rock, Department of Information Science, 2801 S University Avenue, Little Rock, AR 72204, U.S.A.; email: xwxu@ualr.edu.

https://doi.org/10.1145/3068335

1 INTRODUCTION

DBSCAN [16] published at the KDD'96 data mining conference is a popular density-based clustering algorithm which has been implemented independently several times and is available in clustering toolkits such as ELKI [41], scikit-learn [37], R [44], Weka [20], and many others. It is discussed in textbooks [20, 21, 43] and was successfully used in many real-world applications. Many density-based methods were inspired or based on the ideas published in DBSCAN and try to improve upon its limitations [3, 11, 13, 24]. It is an algorithm proven to work in practice, and in 2014, it received the SIGKDD test-of-time award.

The recently published paper [18] presents an interesting complexity analysis of the DBSCAN problem with Euclidean distance, and a new method for this special case. Unfortunately, however, the paper also contains exaggerated and misleading phrases and statements, as well as a biased evaluation of the proposed method, raising perhaps false expectations about the practical performance of the proposed method (which only works with Euclidean distance and point data, and focuses on extreme, and often unsuitable values of the ε parameter), compared to the original DBSCAN algorithm with indexes such as the R*-tree, k-d tree, or cover tree.

Because of the attention that this SIGMOD paper has received, we think it is necessary to clarify these issues in the current technical correspondence, to avoid further confusion about DBSCAN in the database community.

The remainder of this article is organized as follows: In Section 2, we first give an introduction to the DBSCAN algorithm, some related work, and highlight some of its properties. In Section 3, we then discuss the theoretical contribution of the SIGMOD 2015 paper in the context of previous statements about the complexity of DBSCAN and results from computational geometry and indexing. In Section 4, we investigate the methods proposed in the SIGMOD 2015 paper and their performance, by providing first some recommendations for using DBSCAN and giving indicators when DBSCAN parameters may have been chosen poorly, followed by our experiments with the proposed methods, the results of which suggest a different conclusion than the SIGMOD 2015 paper. Section 5 concludes with final remarks.

2 BACKGROUND

The DBSCAN [16] article consists of two major contributions. The first contribution is a formal model for density-based clusters; the second contribution is a database-oriented algorithm to find clusters that adhere to this model, which benefits from advances in database indexing.

2.1 DBSCAN Cluster Model

The model introduced by DBSCAN uses a simple minimum density level estimation, based on a threshold for the number of neighbors, *minPts*, within the radius ε (with an arbitrary distance measure). Objects with more than *minPts* neighbors within this radius (including the query point) are considered to be a *core point*. The intuition of DBSCAN is to find those areas, which satisfy this minimum density, and which are separated by areas of lower density. For efficiency reasons, DBSCAN does not perform density estimation in-between points. Instead, all neighbors within the ε radius of a core point are considered to be part of the same cluster as the core point (called *direct density reachable*). If any of these neighbors is again a core point, their neighborhoods are transitively included (*density reachable*). Non-core points in this set are called *border points*, and all points within the same set are *density connected*. Points which are not density reachable from any core point are considered *noise* and do not belong to any cluster.

Figure 1 illustrates the concepts of DBSCAN. The *minPts* parameter is 4, and the ε radius is indicated by the circles. N is a noise point, A is a core point, and points B and C are border points.

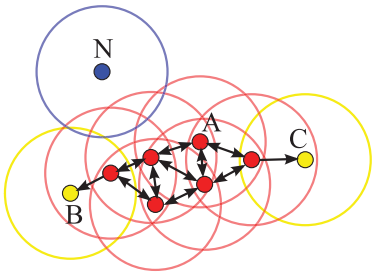


Fig. 1. Illustration of the DBSCAN cluster model.

Arrows indicate direct density reachability. Points B and C are density connected, because both are density reachable from A. N is not density reachable, and thus considered to be a noise point.

In HDBSCAN* [13], the concept of *border points* was abandoned, and only core points are considered to be part of a cluster at any time, which is more consistent with the concepts of a density level set. OPTICS [3], LSDBC [11], and HDBSCAN* are examples of DBSCAN variants that focus on finding hierarchical clustering results. In the more general model of kernel density estimation, DBSCAN uses the simple uniform kernel, with bandwidth $h = \varepsilon$ and a density threshold of $minPts/n$. DenClue [24] is an example of a method that considers alternative kernels for density estimation.

2.2 DBSCAN Clustering Algorithm

The second contribution of DBSCAN is an algorithm to compute clusters according to the above model (except that border points belonging to multiple clusters are only assigned to one of them). In this algorithm, the database is linearly scanned for objects which have not yet been processed. Non-core points are assigned to noise, and when a core point is discovered, its neighbors are iteratively expanded and added to the cluster. Objects that have been assigned to a cluster will then be skipped when encountered later by the linear scan. This basic algorithm is the standard approach to compute the transitive closure of a relation, with the minimal modification that only core points are expanded. Yet, this can yield a reasonably efficient algorithm if a database index is used. Algorithm 1 gives a simplified pseudo-code for this DBSCAN algorithm. There are two calls to the function RangeQuery in this pseudocode. Both line 3 and line 13 will only execute if the point has not yet been labeled, in which case the point is subsequently labeled. This makes it easy to see the following properties: (i) Neighborhood queries are only executed for points labeled *undefined*. (ii) If a neighborhood query is executed on a point, the point is subsequently labeled either with a cluster label or *Noise*. (iii) The only time a point is relabeled is when its label changes from *Noise* to a cluster label in line 11. Thus, we execute exactly one neighborhood query for every point in the database. If the seed set iteration in line 10 is implemented adequately, this loop executes at most once for every point in the database, and we get a runtime complexity of $O(n \cdot Q + \sum_i r_i)$ where Q is the complexity of the function RangeQuery and r_i is the result size of the ith query. If the data is not already stored in an indexed database, we may need to take the index construction time C into account, which yields a total runtime complexity of $O(C + n \cdot Q + \sum_i r_i)$. The index may also need additional storage, so we have space complexity $O(n + I)$ to store the cluster labels and the index. The index may, however, be shared across multiple runs of the algorithm, and different algorithms can benefit from the same index, if it is a "general purpose" index (such as an R*-tree) that supports range queries (among other queries). Implementing the range query with a linear scan yields $Q \in \Theta(n \cdot D)$ with cost D of computing the distance of two points, and then the DBSCAN runtime complexity is $\Theta(n^2 \cdot D)$. For Euclidean space of dimensionality d, this is $\Theta(n^2 \cdot d)$, but when using

ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: DB: Database
Input: ε: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Data: label: Point labels, initially undefined
foreach point p in database DB do                    // Iterate over every point
    if label(p) ≠ undefined then continue            // Skip processed points
    Neighbors N ← RangeQuery(DB, dist, p, ε)         // Find initial neighbors
    if |N| < minPts then                             // Non-core points are noise
        label(p) ← Noise
        continue
    c ← next cluster label                           // Start a new cluster
    label(p) ← c
    Seed set S ← N \ {p}                             // Expand neighborhood
    foreach q in S do
        if label(q) = Noise then label(q) ← c
        if label(q) ≠ undefined then continue
        Neighbors N ← RangeQuery(DB, dist, q, ε)
        label(q) ← c
        if |N| < minPts then continue                // Core-point check
        S ← S ∪ N
```

expensive distance functions such as the shared-nearest-neighbors distance [25] the complexity can even be $O(n^3)$. DBSCAN never has been constrained to using Euclidean distance, or to points in $\mathbb{R}^d$, but has always been intended to also be used with geographic data, polygons, and other data types, as seen in Ester et al. [16], Sander et al. [40], and Xu et al. [47].

This "original DBSCAN" algorithm is not the only algorithm to produce a clustering based on the theoretical DBSCAN cluster model. For example, scikit-learn 0.16 [37] includes a variant that first materializes all neighborhoods (which yields worst-case quadratic memory), then performs the cluster expansion in a "vectorized" way on the core points only. The overall runtime complexity is not improved, but this is more efficient to execute by the Python/NumPy runtime environment. The DBSCAN algorithm can easily be abstracted to the procedure given in Algorithm 2. Tan et al. [43] include a similar pseudocode in their textbook.

The result of this DBSCAN algorithm is deterministic, but may change if the dataset is permuted. Firstly, cluster labels can trivially change depending on the order in which the clusters are discovered. Secondly, border points in the DBSCAN theoretical model can be density-reachable from more than one cluster. The original DBSCAN algorithm simply assigns border points to the first cluster they are reachable from, because a unique cluster assignment is often desirable from a user point of view and this requires the least amount of memory.[1] Because this is a rare situation of little interest, it is not necessary to try different permutations of the dataset. In the improved cluster model of HDBSCAN*, this anomaly has been removed since there are no border points anymore.

[1]Note that this can, in rare cases, lead to a cluster with fewer than *minPts* points, if too many border points are reachable by different clusters, and have previously been assigned to other clusters. Every cluster will at least have one core point. Multi-assignment to exactly represent the theoretical model—or an assignment by shortest distance—can be implemented easily.

ALGORITHM 2: Abstract DBSCAN Algorithm

```
1 Compute neighbors of each point and identify core points        // Identify core points
2 Join neighboring core points into clusters                      // Assign core points
3 foreach non-core point do
4 |   Add to a neighboring core point if possible                  // Assign border points
5 |   Otherwise, add to noise                                      // Assign noise points
```

As we have seen, the essential question to discussing the complexity of the original DBSCAN algorithm is the runtime complexity Q of the neighborhood query RangeQuery. This query constitutes a reusable (other algorithms can also use RangeQuery) and powerful interface for implementing and optimizing data mining algorithms efficiently on top of a database. But if this query is not accelerated, the runtime is $\Theta(n^2 \cdot D)$ with D being the cost of a distance computation.

3 DISCUSSION OF THE SIGMOD 2015 ARTICLE

Gan and Tao [18] make the following main contributions:

(1) A reduction from the USEC problem to the Euclidean DBSCAN problem, i.e., from USEC to the problem of finding the density-based clusters, according to the definitions introduced in Ester et al. [16], in a set of d-dimensional points when $d \geq 3$ and using Euclidean distance. This reduction leads to a lower bound of $\Omega(n^{4/3})$ for the time complexity of every algorithm that solves the Euclidean DBSCAN problem in $d \geq 3$.
(2) A proposal for an approximate, grid-based algorithm for density-based clustering of d-dimensional points using Euclidean distance with theoretical guarantees for the cluster approximation quality.

The article also brings to the attention of the database community that a worst-case runtime complexity of $O(n \log n)$ for the DBSCAN algorithm, when supported by index structures, cannot be correct, given the theoretical results. Gan and Tao's observation is correct that this unproven runtime complexity for DBSCAN with index structures has been repeatedly stated in the literature, and that it originates from a statement in the original DBSCAN paper by Ester et al. [16]. The article by Gan and Tao will hopefully lead to a more careful discussion of DBSCAN's runtime complexity in literature, and a more thorough discussion of query complexity of neighbor search in general.

3.1 A Critique of Some Statements in the SIGMOD 2015 Article

We have no issue with the core contributions of the article by Gan and Tao [18], and we fully support the effort to remedy the propagation of the incorrect runtime complexity of DBSCAN with index structures. However, we think the article would have benefited from inclusion of more careful analyses and resulting statements about DBSCAN. We proceed to revisit four statements that we find inaccurate.

It is correct that the DBSCAN paper comes—in a loose manner—to the incorrect conclusion that "small" range queries can be answered with an R*-tree in $O(\log n)$, on average. This incorrect assumption about R*-trees is then used to derive an average runtime complexity of the DBSCAN algorithm when using an R*-tree as $O(n \cdot \log n)$. In the original publication, Ester et al. [16] stated:

> Region queries can be supported efficiently by spatial access methods such as R*-trees [6] which are assumed to be available in a SDBS for efficient processing of several types of spatial queries [12]. The height of an R*-tree is $O(\log n)$ for a database of n points in the worst case and a query with a "small" query region has to traverse only a limited number of paths in the R*-tree. Since the

> Eps-Neighborhoods are expected to be small compared to the size of the whole data space, the average run time complexity of a single region query is $O(\log n)$. For each of the points of the database, we have at most one region query. Thus, the average run time complexity of DBSCAN is $O(n \cdot \log n)$.

These statements about the runtime of DBSCAN in the original paper are clearly not a formal complexity analysis, but are informal and are not part of the main contribution of the paper. In retrospect, the runtime of DBSCAN with an R*-tree should have been discussed more carefully, without relying on the $O(\log n)$ complexity for the runtime of range queries, thus avoiding a formal statement of $O(n \cdot \log n)$ as the average runtime of DBSCAN with index structures. While it has been observed that range queries can be supported efficiently using R*-trees in a wide range of applications and datasets, no formal analysis of the average runtime complexity has yet been conducted in the literature (see Section 3.2 for a more detailed discussion).

The way this error is presented in Gan and Tao [18], however, paints an inaccurate picture of what is and what is not known about density-based clustering and DBSCAN. On page 1, Gan and Tao [18] write in the first paragraph of Section 1.1., which has the title "A Mis-Claim of 17 Years":

> 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 [19] recently, the algorithm of [16] actually runs in $O(n^2)$ worst case time, regardless of the parameters ρ and MinPts.

There are several problems with this way of presenting the mistake: Firstly, Ester et al. [16] do not make a claim about a worst-case performance guarantee, but derive informally (making a wrong assumption about the runtime complexity of range queries in R*-trees) an *average* runtime complexity, in the intended class of applications, which was only informally characterized as cases where range queries are "small"—note the quotation marks around "small" in the original text, which (unsuccessfully) tried to indicate that not just the range but also the result size is assumed to be small. Secondly, the worst-case quadratic runtime of DBSCAN can be found in numerous sources, and was not just "recently" pointed out "after 17 years" in the masters thesis by Gunawan [19]. In fact, there is no indication that Gunawan considers the $O(n^2)$ worst-case runtime of DBSCAN as as a novel insight. He writes [19]:

> The running time of DBSCAN depends on how many times the function RangeQuery(p, ε) is called. We can see that DBSCAN calls this function exactly once for each point. Therefore, the total running time complexity for DBSCAN is $O(n^2)$ using the most naive way to do a range query …

This is almost the same formulation as in Sander et al. [40], which states regarding the worst case (which is the case when indexes cannot efficiently support range queries):

> The runtime of GDBSCAN obviously is $O(n \cdot$ runtime of a neighborhood query$)$. n objects are visited and exactly one neighborhood query is performed for each of them. […] Without any index support, the runtime of GDBSCAN is $O(n^2)$.

Gunawan [19] also represents the mistake in the KDD'96 paper more correctly as a mistake in the runtime complexity assumptions of range queries with index structures:

> In [16], it is claimed that range queries can be supported efficiently by using spacial [sic] access methods such as R*-tree. By using it, the average running time complexity can be reduced to $O(n \log n)$. They claimed this to be true because the height of the tree is in $O(\log n)$, and a query with a small query range (such

> as ε-neighborhood) has to traverse only a limited number of paths in the tree. However, there is no theoretical guarantee that this holds . . .

Most secondary sources are clearly aware of the worst-case complexity but, unfortunately, often also re-state the complexity of DBSCAN when using index structures as $O(n \cdot \log n)$—sometimes even without qualifying it further. This runtime complexity is formally incorrect since there is no theoretical guarantee for the assumed runtime complexity of range queries, and it should not be repeated in this form. This mistake in the stated runtime complexity, however, does not mean that there is an un-fixable flaw in DBSCAN, nor does it mean that DBSCAN cannot be efficient with index structures in many clustering applications.

The second problematic aspect of Gan and Tao [18] is the language used when portraying the runtime complexity of DBSCAN. Emotional appeals such as in "indicates (sadly) that all DBSCAN algorithms must be intolerably slow even on moderately large n in practice" [18] are imprecise and are better left out. Further, the obtained complexity result of $\Omega(n^{4/3})$ in $d \geq 3$ hardly justifies statements like "DBSCAN is computationally intractable in practice, even for moderately large n!". Calling even the worst-case complexity of $O(n^2)$ "intractable" is a stretch of the terminology as used in theoretical computer science. Even a linear-scan implementation of DBSCAN with a runtime complexity $\Theta(n^2)$ can be useful for data clustering. And it is also still the case that index structures do speed up DBSCAN significantly in many practical applications, even though the class of applications in which one can observe a logarithmic runtime of small range queries has not been formally characterized.

The third problematic aspect of Gan and Tao [18] is that it constrains DBSCAN to Euclidean distance (Section 2.1), while the original publication clearly states "our approach works with any distance function so that an appropriate function can be chosen for some given application" [16]. All claims and proofs of Gan and Tao [18] implicitly assume the use of the Euclidean distance, and the proposed new algorithms cannot be used with arbitrary distances; i.e., they only work for a subset of DBSCAN problems.

The fourth problematic aspect of Gan and Tao [18] is its experimental evaluation of the proposed method, which does not support the claim that ρ-approximate DBSCAN "should replace DBSCAN on big data due to the latter's computational intractability." We have dedicated Section 4 to experiments with the implementation of the SIGMOD paper.

3.2 The Relationship Between the DBSCAN Algorithm and Range Queries

As one of the anonymous reviewers of an earlier draft of this article pointed out: "For a 'database person.' the relevant kernel of DBSCAN is quite familiar: it's a self-join [. . .] with distance predicate. This can be implemented via a $O(n^2)$ nested loops join. Or it can perhaps be accelerated by an 'index join'—e.g., by use of an R*-tree on the 'inner relation [. . .]. It should be clear that the performance depends on the effectiveness of the index [. . . and] that the relevant theory relates to indexing." The KDD'96 paper did not make a claim about the hardness of the DBSCAN *problem*, but derived a (wrong) average runtime complexity of the proposed DBSCAN *algorithm*, based on the wrong assumption of $O(\log n)$ for the average runtime of range queries (on which this particular algorithm is based). Credit has to be given to Gan and Tao [18] for showing a worst-case complexity of $\Omega(n^{4/3})$ in $d \geq 3$ for the Euclidean DBSCAN *problem*, based on a reduction from the USEC problem to the DBSCAN problem. This shows that the derived bound applies to any algorithm that computes density-based clusters according to the cluster model proposed in Ester et al. [16]. This general result was not known at the time. Known at the time were results though that directly relate to the complexity of the original DBSCAN algorithm. Particularly, Erickson [15] showed a similar lower bound for a wide class of geometric problems, including the following:

> Detecting, counting, or enumerating incidences between a set of "point-like" geometric objects (points, line segments, circles, triangles, etc.) and a set of "line-like" geometric objects (lines, line segments, rays, circles, etc.)

Any form of neighborhood or connectedness (e.g., density-connectedness and single/linkage clustering) is part of this problem type, for which Erickson [15] noted: "We leave the reductions as easy exercises for the reader" and gives a few examples for "range searching" problems.

Matousek [35] noted "algorithms of this type [...] can be used as subroutines in solutions to many seemingly unrelated problems, which are often motivated by practical applications." The original DBSCAN algorithm is exactly such a practical application using RangeQuery as a subroutine.

If we consider the relationship between RangeQuery and the unit-spherical emptiness checking (USEC) problem,[2] and differentiate between query and construction time, similar bounds for the original DBSCAN algorithm (because it uses RangeQuery) follow easily:

Theorem 3.1. *Let $D \subset \mathbb{R}^d$ ($d \geq 3$) be a database containing $|D| = n$ points in $\mathbb{R}^d$, with dimensionality d and Euclidean distance.*

(*i*) *The USEC problem can be solved by ε-range queries.*
(*ii*) *If the USEC problem is in $\Omega(n^{4/3})$ (Hopcroft hard), then finding the ε-range neighbors is at least $\Omega(n^{1/3})$, or preprocessing is at least $\Omega(n^{4/3})$.*

Proof. (i) USEC can be solved by storing the point set in a database, and for every ball performing a RangeQuery with radius $\varepsilon = 1$ (unit) on the database. The answer to USEC is "non-empty" exactly if some query returns a match, and "empty" otherwise. (ii) If finding the ε-range neighbors were faster than $\Omega(n^{1/3})$, doing a repeated ε-range neighbors search as in (i) would yield a faster USEC algorithm unless preprocessing requires $\Omega(n^{4/3})$. □

It is easy to see from the proof of (i) that the USEC problem is closely related to the RangeQuery problem, and this limitation thus affects any algorithm that performs range queries for every point.

To get deeper insight into this problem, we can also look at the bichromatic closest pairs (BCP) problem[3] instead of USEC. The BCP problem is (for dimensionality $d \geq 3$) "believed to be roughly $\Theta(n^{4/3})$" [15], and is another one of the Hopcroft hard problems discussed by Gan and Tao. We can easily obtain the following similar result:

Theorem 3.2. *Let $D \subset \mathbb{R}^d$ ($d \geq 3$) be a database containing $|D| = n$ points in $\mathbb{R}^d$, with dimensionality d and Euclidean distance.*

(*i*) *The BCP problem can be solved via nearest-neighbor queries.*
(*ii*) *If the BCP problem is in $\Omega(n^{4/3})$ (Hopcroft hard), then finding the nearest neighbor is at least $\Omega(n^{1/3})$, or preprocessing is at least $\Omega(n^{4/3})$.*

Proof. (i) BCP can be solved by finding the nearest neighbor in the first ("red") set for each object in the second ("blue") set, and simply returning the pair with the smallest distance. (ii) If finding the nearest neighbor were faster than $\Omega(n^{1/3})$, doing a repeated nearest-neighbor search as in (i) would yield a faster BCP algorithm unless preprocessing requires $\Omega(n^{4/3})$. □

Remark 3.3. For $d = 2$, USEC and BCP are not in $\Omega(n^{4/3})$, thus (ii) in either theorem is ex falso quodlibet and the constraint $d \geq 3$ in the theorems is not required.

[2] In the USEC problem, we have both a set of points and a set of balls with equal radius in $\mathbb{R}^d$, for a constant d, and we have to determine whether at least one point is covered by some ball.
[3] In the BCP problem, we have two sets of points ("red" and "blue"), that we can intuitively consider as data points and query points, and where we need to find the closest red-blue pair (i.e., the nearest neighbor to any query point).

The consequence of Theorems 3.1 and 3.2 is that *there exists no universal index* which can guarantee $O(\log n)$ time for nearest-neighbor search or ε-range neighbor search for *arbitrary* data and which can be constructed in $O(n \log n)$. For any index, there must be at least one data set where the points are distributed in such a way that finding the nearest neighbor takes at least $\Omega(n^{1/3})$ time, unless expensive preprocessing is used or $d \leq 2$.

At the time the KDD'96 paper was written, these results had not yet permeated deeply into the data mining community (and one may argue, they still have not). Matousek [34] already noted that range-searching algorithms are "one of the central themes in computational geometry." In the context of data indexing, the relationship of k-nearest-neighbor queries to the ball-emptiness problem (and thus USEC) was already recognized, for example, by Arya and Mount [5], and a motivation for their approximate nearest-neighbor search algorithm.

While the above results rely on complexity assumptions for the Hopcroft problems, these are widely believed to be true. The proof relies on Euclidean distance, but this will likely transfer to many other distance measures (in particular, it must transfer to distances based on the kernel trick and scalar products in finite vector spaces; but it does not transfer to the discrete metric). In contrast to earlier—theoretical and empirical—results on non-indexability due to high dimensionality [8, 25, 46, 48], this result indicates that bad data distributions are possible in as little as three to five dimensions where the curse of dimensionality is not yet considered to be problematic. But even in these cases, $\Omega(n^{4/3})$ would still allow a considerable possible speedup over the linear scan of $\Theta(n^2)$.

Regarding the complexity of indexing schemes for general database workloads, Hellerstein et al. [22] and Samoladas and Miranker [39] provide some lower bounds. In the more specific context of R-trees, the question of the query complexity of spatial queries is not yet ultimately answered. Faloutsos and Kamel [17] and Papadopoulos and Manolopoulos [36] discuss the R-tree query performance, but with strong assumptions on the distribution of the data and the structure of the tree. They focused on estimating query selectivity, which allows a database system to decide when to use the index, and when to use a linear scan. Manolopoulos et al. [33] survey further results. Priority-R-trees have been shown to answer window queries in $O(n^{1-1/d} + r)$ by Arge et al. [4]. But despite the limited theoretical insights into the query complexity of R-trees and similar indexes, they have proven to work well in many practical applications. And a theoretically superior algorithm may be slower in practical use because of large constant factors neglected in theoretical complexity analysis.

We would like to point out that it is common to see new indexes claim $O(\log n)$ nearest-neighbor query time. For example, Beygelzimer et al. [10] introduce the cover tree and claim $O(\log n)$ query and $O(n \cdot \log n)$ construction time, and attribute the same to Karger and Ruhl [27], Hildrum et al. [23], and navigating nets [28] with different constant factors. But these results probably do not contradict Theorem 3.2, because they make the assumption of a growth restricted metric: "The complexity of these operations depends on the dimension of S and the aspect ratio of (S, d), denoted $\Delta = \Delta_S$, which is defined as the ratio between the largest and smallest interpoint distances in S. In most applications, $\Delta = \text{poly}(|S|)$, hence $\log \Delta = O(\log n)$" [28]. The use of "most applications" suggests that this is only a typical behavior and not a worst-case result for arbitrary datasets. Similarly, Arya and Mount [5] claimed "the first algorithm for nearest neighbor searching (approximate or exact) that achieves both polylogarithmic search time and nearly linear space." This does not contradict Theorem 3.2, because they require $O(n^2)$ construction time.

In conclusion, Gan and Tao [18] showed that no algorithm for computing density-based clusters according to the DBSCAN model can run in time of $o(n^{4/3})$. This bound was only known for problems that require n range queries (as the original DBSCAN algorithm does), but has not found the needed attention in the database community. However, this bound hardly means that DBSCAN is "intractable in practice," and even though logarithmic runtime of range queries in

index structures cannot be guaranteed, in practical applications of clustering (where ε is typically small and queries have small result sizes), a substantial speedup of range queries with index structures can be observed regularly. As an orthogonal approach to improve the efficiency of DBSCAN, recent work by Mai et al. [32] proposes a method that can achieve a substantial reduction of the number of range queries necessary for DBSCAN. This method can, in many applications, lead to speedups of orders of magnitude compared to the original DBSCAN algorithm.

3.3 Grid-Based Approaches and Approximations of DBSCAN

The idea of approximating results for better performance, as well as grid-based approaches to solve the DBSCAN problem have been around for a long time. When runtime is crucial, the inaccuracies of approximations may be well within the tolerance limits of the application, and approximative similarity search using k-d-trees [5], or locality sensitive hashing (LSH) [26] are as old as DBSCAN, and have successfully been combined with DBSCAN. Grid-based approaches such as STING [45], GriDBSCAN [31], and the new approach by Gan and Tao [18], which is the second main contribution in the article by Gan and Tao, can (i) quickly identify regions of high density as core points, (ii) process multiple points at once, and (iii) quickly prune regions of low density as noise; which can provide considerable speedup if the grid size is chosen adequately.

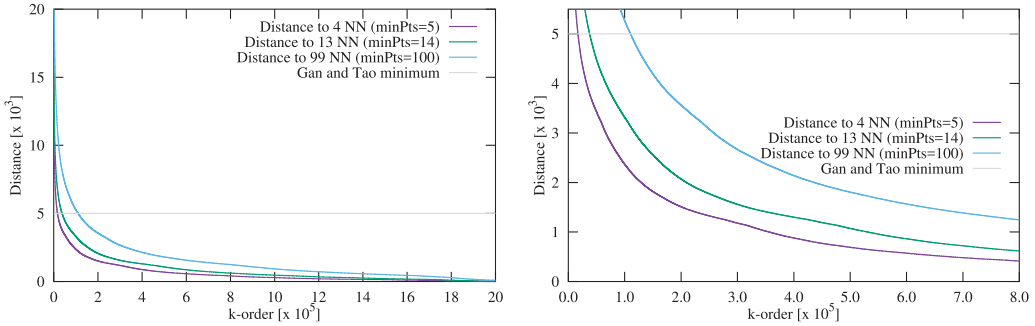
However, the new method by Gan and Tao [18] has limitations that make the claim that the new approximate algorithm "should replace DBSCAN on big data" questionable.

Similar to R*-trees and most index structures, grid-based approaches tend to not scale well with increasing number of dimensions, due to the curse of dimensionality [25, 48]: the number of possible grid cells grows exponentially with the dimensionality: dividing the dataset into b bins in each of d dimensions yields b^d grid cells. The new method is no exception: the total number of cells intersecting the boundary is shown to be $O(1 + (1/\rho)^{d-1})$ [18]—with the suggested value of $\rho = 1/1000$, this means that theoretically $O(1 + 1000^{d-1})$ neighbor cells need to be considered. This issue is not investigated in Gan and Tao [18] that treats the dimensionality as a constant in the analysis and chooses very large values for the parameter ε in the experimental analysis.

Independent of the algorithm, the parameter ε of DBSCAN becomes hard to choose in high-dimensional data due to the loss of contrast in distances [9, 25, 48]. Irrespective of the index, it therefore becomes difficult to use DBSCAN in high-dimensional data because of parameterization; other algorithms such as OPTICS and HDBSCAN* that do not require the ε parameter are easier to use, but still suffer from high dimensionality.

One of the strengths of the original DBSCAN algorithm is that it can be paired with any data type, distance function, and indexing technique adequate for the dataset to be analyzed. Grid-based approaches—including the approach by Gan and Tao [18]—are usually designed for point data and Minkowski distances such as Euclidean distance only. This is a limitation not shared by DBSCAN: The R*-tree can be used both with data such as polygons, as well as with other distance functions. Distances supported by the R*-tree include cosine distance, several divergence measures, and geodetic distance [42] that are important in practice. For text data, inverted indexes are the standard solution. Cover-trees and M-trees can work with any distance that satisfies the triangle inequality and can be used with the DBSCAN algorithm. For practical use, the ability to use the algorithm with other domain specific distance functions (see, e.g., Andrienko et al. [2] and the Encyclopedia of Distances [14]) is important, and grid-based approaches will usually no longer be applicable—although, the design of cleverer grid construction methods for currently not supported distance functions may be possible and could be an interesting direction for future research.

In conclusion, while Gan and Tao [18] claim their approximate algorithm "should replace DBSCAN on big data," it will only be able to do so for the subset of DBSCAN problems that it supports, while the original DBSCAN algorithm applies more generally, and indexing strategies



(a) k-distance plots (b) k-distance plots (magnified region)

Fig. 2. Sorted k-distance plots on the "Household" data.

for other metrics may still be able to accelerate it. When a grid-based approach is applicable, Gan and Tao's approximate algorithm has the theoretical advantage over other grid-based approaches that it comes with a bound on the approximation quality. However, the practical advantage of the algorithm as outlined in Gan and Tao [18] has yet to be established convincingly for smaller and often more useful parameter values.

4 RECOMMENDATIONS FOR DBSCAN AND NEW EXPERIMENTAL EVALUATION

4.1 Heuristics for Choosing DBSCAN Parameters

The easier-to-set parameter of DBSCAN is the *minPts* parameter. Its purpose is to smooth the density estimate, and for many datasets it can be kept at the default value of $minPts = 4$ (for two-dimensional data) [16]. Sander et al. [40] suggest setting it to twice the dataset dimensionality, i.e., $minPts = 2 \cdot \text{dim}$. For datasets that have a lot of noise, that are very large, that are high-dimensional, or that have many duplicates it may improve results to increase *minPts*.

The radius parameter ε is often harder to set. The way DBSCAN is designed, ε should be chosen as small as possible. The value of ε also depends on the distance function. In an ideal scenario, there exists domain knowledge to choose this parameter based on the application domain. For example, when clustering GPS locations, a domain expert may decide that objects within 1 kilometer should be neighbors and we then can choose the appropriate distance function and set ε accordingly. But if we set this parameter based on domain knowledge, we may want to vary *minPts*. It is desirable to have at least one free parameter to vary, such that we can get different insights into our data.

Ester et al. [16] provide a heuristic for choosing the ε parameter in Section 4.2 based on the distance to the fourth nearest neighbor (for two/dimensional data). In Generalized DBSCAN, Sander et al. [40] suggested using the $(2 \cdot \text{dim} - 1)$ nearest neighbor[4] and $minPts = 2 \cdot \text{dim}$.

We applied these heuristics to the Household dataset from the UCI repository [30], as obtained from Gan and Tao [18], who removed missing values, and normalized the data to integers in $[0; 10^5]$. This dataset has 5% duplicate records, and attributes have only 32 to 4,186 different values each, so it has a low numerical resolution. This causes artifacts (visible steps) in some of the plots due to tied distances. In Figure 2, we plot the sorted k-dist graph as discussed in the original DBSCAN paper [16, Section 4.2], which is a plot of the k-nearest-neighbor distances, computed for each point, and plotted from largest to smallest value. We use the values $k = 4$, $k = 13$ as obtained by the above heuristics, and $k = 99$ so that $k + 1 = 100 = minPts$ as used by Gan and Tao.

[4]The k-nearest neighbors do *not* include the query point, but the RangeQuery does include the query point for density estimation. Therefore, k corresponds to $minPts = k + 1$ and thus $minPts = 2 \cdot \text{dim}$.

At first sight, there is no clearly visible "valley," "knee," or "elbow" in this plot, as seen in Figure 2(a). When zooming into the interesting region in Figure 2(b), it seems most appropriate to set the cut somewhere between $\varepsilon \approx 500$ and $\varepsilon \approx 2{,}000$.[5] Experience shows that typically the lower values work better, e.g., $\varepsilon \leq 1{,}000$.

Interestingly, these results are rather stable across the choice of *minPts*, supporting the observation that *minPts* often has little impact on the clustering results and they do "not significantly differ" [16, Section 4.2]. The minimum ε used by Gan and Tao, however, is clearly in a region where the values are still steeply dropping, and by the heuristics suggested for DBSCAN, the value should clearly be chosen below $\varepsilon \ll 2{,}000$ instead of only using $\varepsilon \geq 5{,}000$ as used in Gan and Tao [18].

4.2 Red Flags for Degenerate Clustering Results

In the following, we discuss two very basic "red flag" criteria for detecting degenerate DBSCAN results: the first is the number of clustered points (or, equivalently, the number of noise points), and the second is the size of the largest component. We will also measure the number of clusters.

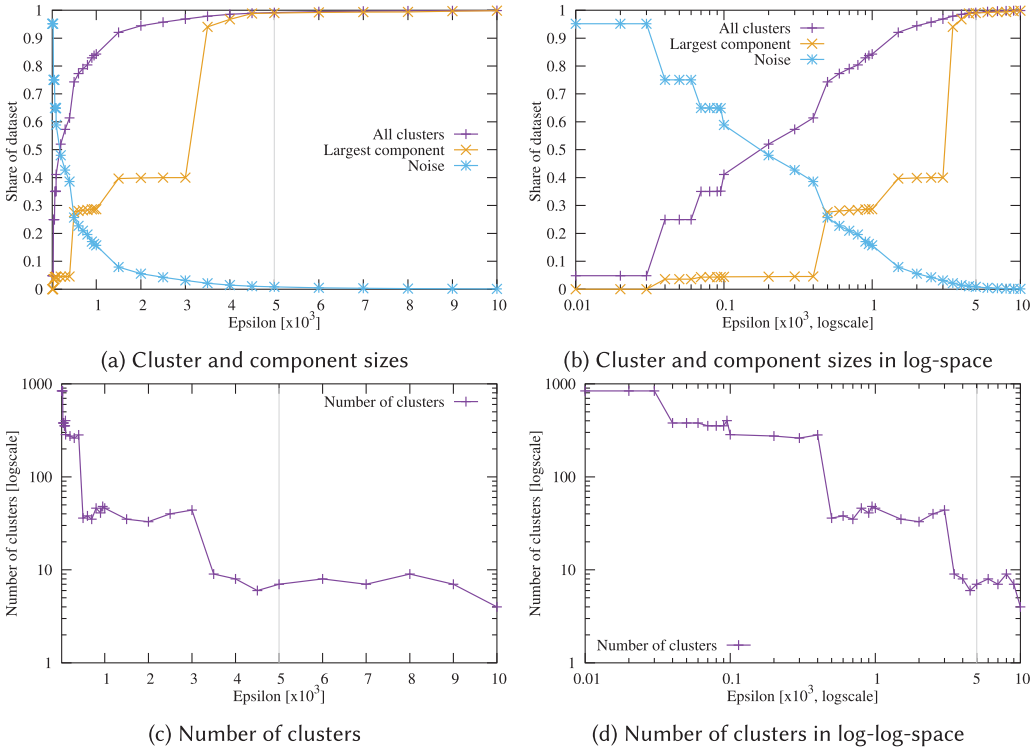
An interesting clustering result—one that can provide insight to the practitioner—should include a large part of the dataset, but a key feature of DBSCAN that sets it apart from k-means is that it can also detect noise. The desirable amount of noise will usually be between 1% and 30%. The size of the largest component (i.e., the largest cluster) can also be used to detect if a clustering has degenerated. In the context of DBSCAN, a largest component that includes most of the data indicates that ε was chosen too large, or that a hierarchical analysis may be necessary: often there is structure hidden in this component that becomes visible if a smaller ε radius is chosen. If the largest component contains more than 20% to 50% of the clustered points, either a hierarchical approach such as OPTICS [3], or HDBSCAN* [13] should be used, or DBSCAN should be run again with a smaller ε radius.

Neither of these criteria is good to *quantify* the quality of a clustering, but they can serve as a guideline to easily detect degenerate solutions that are usually not useful. Often these observations can be used to guide parameterization, since, e.g., increasing the ε parameter will increase the number of clustered points (and the size of the largest cluster), and often decreases the number of clusters. Conversely, decreasing it will produce more noise, and smaller clusters.

In Figure 3(a) (and Figure 3(b), in logscale) we show the number of clustered points and the size of the largest component. It can be clearly seen that at $\varepsilon = 5{,}000$—the *minimum* value used by Gan and Tao—almost the entire dataset (>99%) is contained in a single cluster. In Figure 3(c) (and Figure 3(d), in logscale) we can observe the number of clusters to suddenly drop at $\varepsilon \approx 500$ and $\varepsilon \approx 3{,}500$, while at the same time, the size of the largest component for the Household dataset increases from 5% to 30% and from 40% to 95% of the dataset, respectively. Obviously some larger structural change happens at these distances. We manually analyzed what is happening at these distances, and this revealed that these changes are artifacts resulting from the low resolution of the dataset, which contains seven attributes with a low numerical precision, and are not particularly interesting: because attributes have as little as 32 different values, we observe many identical distances and increasing the ε threshold suddenly connects many more points than before. The way the dataset has been preprocessed may be inadequate to solve a real problem on this dataset.

The heuristic of keeping the amount of noise between 1% (assuming clean data and no duplicates) and 30% (assuming noisy data or many duplicates) suggests an appropriate range of $\varepsilon < 500$. The largest connected component heuristic indicates that $\varepsilon \leq 400$ is desirable, and beyond 3,000 the results have degenerated to a single cluster containing most of the dataset, and some small "outlier clusters." Based on these heuristics, the interesting parameter range is $\varepsilon = 100$ to $\varepsilon = 500$

[5] The discrete steps visible in Figure 2(b) below $\varepsilon = 1{,}000$ are due to the low numerical precision of this dataset.



(a) Cluster and component sizes

(b) Cluster and component sizes in log-space

(c) Number of clusters

(d) Number of clusters in log-log-space

Fig. 3. Statistics on DBSCAN clustering results on the "Household" dataset. The vertical lines indicate the minimum ε = 5,000 used by Gan and Tao.

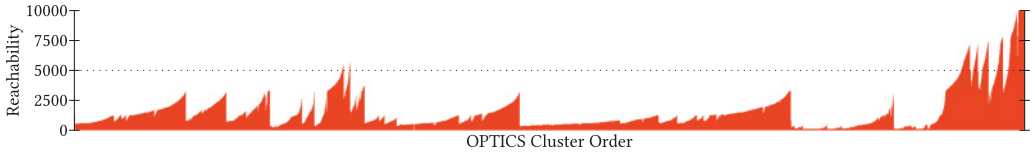


Fig. 4. OPTICS plot for "Household" dataset, with a maximum distance of 10,000.

for the Household dataset. (See Figure 6(c) for the "Farm" dataset and Figure 8(b) for the "PAMAP2" dataset.)

OPTICS [3] plots are a useful tool for understanding the density-based clustering structure (but unfortunately this visual approach does not scale well to large datasets). In Figure 4 we show a downsampled OPTICS plot for the "household" dataset. At $\varepsilon \approx 2{,}500$, most of the valleys disappear. There is some smaller structure recognizable in the plot that requires $\varepsilon < 1{,}500$.

4.3 Relevance Concerns with the Experimental Analysis by Gan and Tao

In the light of such experiences with using and parameterizing DBSCAN, we were surprised by some of the results reported by Gan and Tao [18]: their choice of ε (and *minPts*) was unusually large; and they neither discussed the motivation to choose the parameters this way, nor did they discuss the index used. This problem can, for example, be seen in Figure 9 of Gan and Tao [18], where in Figure 9(e)–(l), ε was chosen too large to properly detect the cluster structure. Without giving

a reason, Gan and Tao decided to choose $\varepsilon = 5{,}000$ as *minimum* value for all their experiments, while we would have used it as a maximum instead.

This parameter choice is beneficial for grid-based approaches because the number of possible grid cells grows exponentially with the dimensionality: e.g., decreasing the value of ε from 5,000 to 1,000 on a seven-dimensional dataset increases the theoretical number of grid cells by a factor of $5^7 = 78{,}125$. Even though much fewer grid cells will be occupied, choosing a small ε value may have a negative impact on the performance of grid-based methods. Figures 12(d)–12(f) in Gan and Tao [18] show that the runtime is decreasing with increasing ε, which is probably due to the decreasing number of grid cells.

4.4 New Runtime Experiments

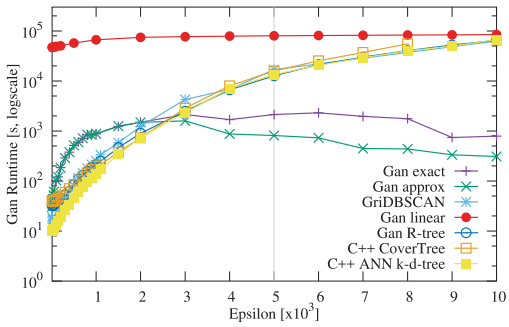
Given these discrepancies with our own experience of using DBSCAN, we decided to reproduce the results of Gan and Tao [18]. While doing fair benchmarks remains a "black art" [29], we try our best to conduct a balanced and fair study. For all experiments we used a Debian Linux PC with 32GB of RAM and an Intel i7-3770 CPU at 3.4GHz (turbo-boost disabled). Gan and Tao readily provided us with their datasets, as well as with a binary C++ implementation. We compare it to the open source Java DBSCAN implementation available in ELKI 0.7.0 [41], which contains implementations of many indexes such as the R*-tree, k-d-tree, and cover tree. Furthermore, we implemented DBSCAN from scratch in C++ using the cover tree code by Beygelzimer et al. [10] and the ANN library by Arya and Mount [5] (which implements the k-d-tree [7] efficiently; we disabled approximative search, and did some small optimizations to their fixed-radius query).

As discussed above, the interesting parameter range is much below the minimum of $\varepsilon = 5{,}000$ used in Gan and Tao [18]. As seen in Figure 5, the use of an R*-tree (or other) indexes can make a huge difference for $\varepsilon < 1{,}000$, at which point we still have some interesting clusters (the largest cluster contains about 29% of the dataset, and 84% of the data has been clustered into 47 clusters). For $\varepsilon < 500$, all the indexes provide a 10- to 1000-fold speedup over the linear scan.

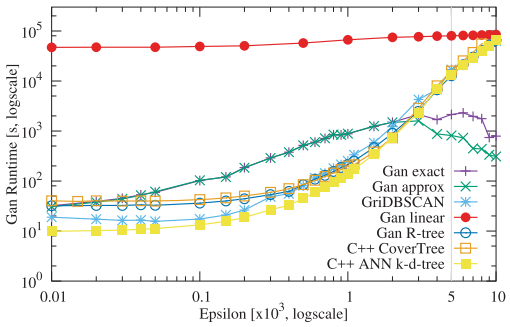
The results using the implementation provided by Gan and Tao—including smaller values of ε—are shown in Figure 5(a) and (b). As expected, all the grid-based approaches work well for very small values of epsilon. The new approaches by Gan and Tao improve over GriDBSCAN by not degrading to a linear scan for large values of $\varepsilon \geq 3{,}000$. But until $\varepsilon = 2{,}000$, their new method is outperformed by their own implementation of GriDBSCAN [31] as well as their implementation of the R-tree. Both our ANN-based and the cover-tree-based C++ implementations perform similarly, demonstrating that all the indexing techniques yield rather comparable speedups. Usually, the ANN k-d-tree was the fastest implementation, likely because this library is very well optimized.

Comparing the runtimes of Gan and Tao's and our C++ implementations to the Java runtimes of ELKI is not fair but nevertheless interesting. It is unfair, because the C++ code was written for this single use case (Gan and Tao even use an integer data representation), and is well optimized by the compiler, whereas ELKI is implemented in Java, uses double precision data, and supports many different distance functions and indexes, features important for practical applicability. The performance of ELKI may thus be more indicative of what we can expect from a general-purpose tool in practical use, such as a modern database system.

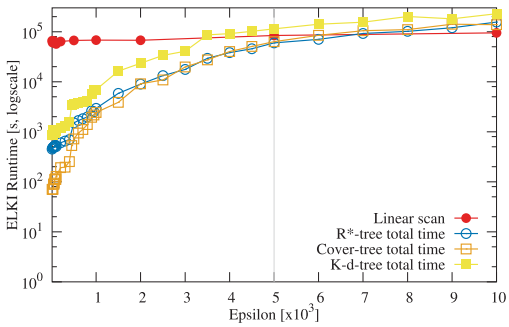
Results on the "Farm" and "PAMAP2" [38] datasets are similar, and are shown in Figure 6 and Figure 8. Again, we can clearly see that the ε parameter was chosen much too large by Gan and Tao: almost all the objects are put into the same cluster at $\varepsilon \geq 5{,}000$. Figure 6(c) shows that the most interesting radii for the "Farm" dataset are $\varepsilon \in [400; 1{,}000]$. By the time the minimum $\varepsilon = 5{,}000$ of Gan and Tao is reached, almost every point is in the same cluster, and only a few "outlier clusters" remain separate. These clusters appear at a much smaller ε; the time when they join the largest



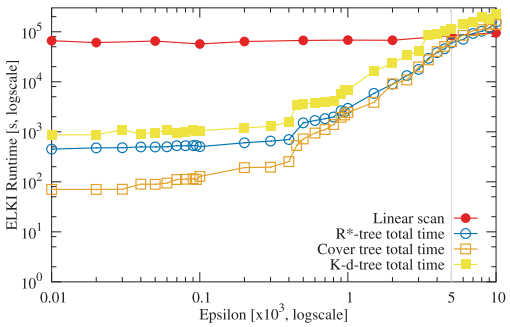
(a) Runtime using the implementations by Gan and Tao, and our C++ versions



(b) Runtime in log-log-space using the implementations by Gan and Tao, and our C++ versions



(c) Runtime using ELKI

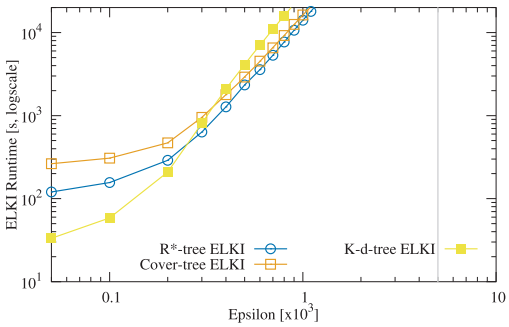


(d) Runtime in log-log-space using ELKI

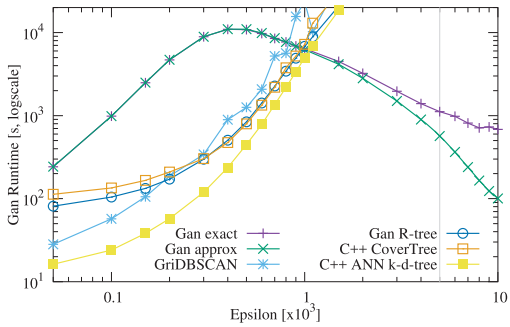
Fig. 5. Runtime performance on the "Household" dataset. The vertical line indicates the minimum $\varepsilon = 5{,}000$ used by Gan and Tao.

component is of much less use. The most interesting value for this dataset is probably around $\varepsilon = 900$, just before the largest component contains almost all the cluster points. On the "Farm" dataset, only few valleys exist in the OPTICS plot. In Figure 7, we marked the most interesting valley, which is probably responsible for the second largest cluster which disappears at $\varepsilon \geq 1{,}000$. Choosing DBSCAN parameters to yield a useful result on such data is very difficult, and this dataset may not contain meaningful density-based clusters.

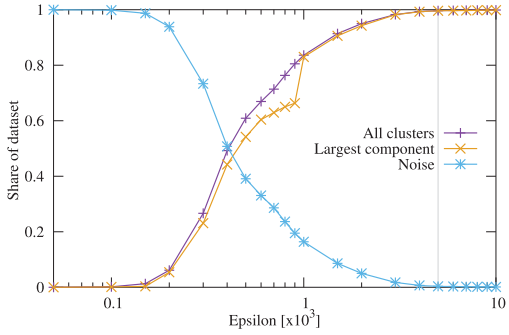
For "PAMAP2" [38], the interesting range is $\varepsilon \in [500; 2{,}000]$ according to Figure 8(b). At $\varepsilon = 5{,}000$ there are two major clusters accounting for 99.6% of the dataset, and no substantial changes in the dataset happen anymore. A scatter plot visualization of a sample (Figure 10) reveals the nature of these two clusters. There clearly is no use in choosing a radius ε large enough to merge them. In Figure 10(c) and (f) we show the clustering result with the minimum $\varepsilon = 5{,}000$ used by Gan and Tao. Most of the data has already merged into the two large components. Choosing a smaller radius such as $\varepsilon = 500$ in Figure 10(a) and (d) yields a much more interesting clustering structure. At $\varepsilon = 1{,}000$ (Figure 10(b) and (e)) we can observe the largest connected component emerge (the yellow cluster in the background, covering much of the bottom group). Judging from these scatter plots, Gan and Tao likely did not remove the timestamp prior to applying PCA. The projected dimension 1 is mostly the timestamps of the data points. The second dimension may capture the overall magnitude of the movements and we suppose that the top "cluster" corresponds to one of the high acceleration activities (running, cycling, rope jumping, car driving), whereas the cluster containing 90% of the data points has the low acceleration activities (sitting, standing, walking,



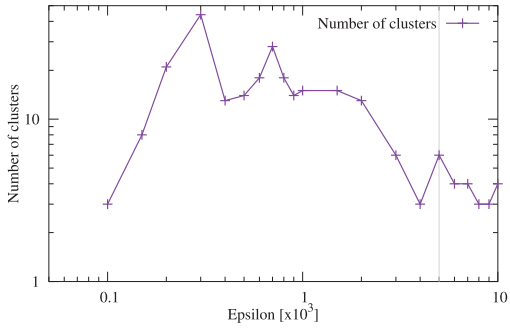
(a) Runtime in log-log-space using the Java implementations from ELKI



(b) Runtime in log-log-space using the implementations by Gan and Tao, and our C++ versions



(c) Cluster sizes in log-log scale



(d) Number of clusters in log-log scale

Fig. 6. Runtime performance on the "Farm" dataset. The vertical line indicates the minimum ε = 5,000 used by Gan and Tao.

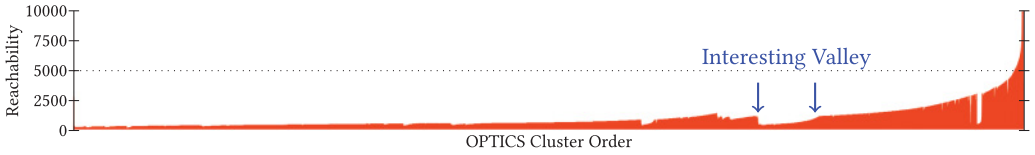


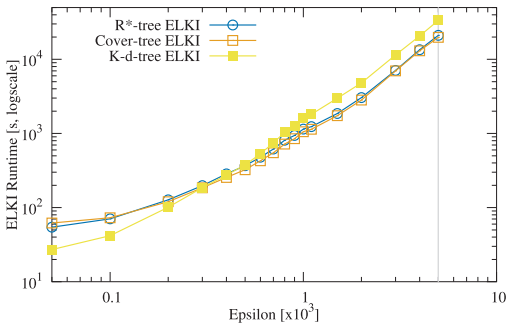
Fig. 7. OPTICS plot for the "Farm" dataset, with a maximum distance of 10,000.

watching TV). Obviously, this dataset should have been analyzed as time series instead; with the original DBSCAN, an appropriate time series distance function could have been used.

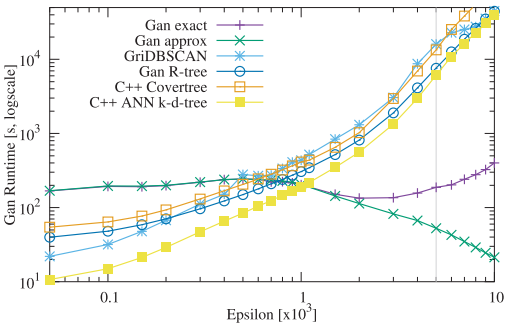
This inappropriate preprocessing also causes the OPTICS plot in Figure 9 to contain two major valleys, separated with a thin "needle" at about 90% of the dataset. We can see the plot to be very detailed, with many small valleys of near-duplicate points. These individual valleys (tiny in the plot, but containing thousands of points because of the dataset size) correspond to the clusters seen in Figure 10(d) and (e). At $\varepsilon \geq$ 5,000, this structure has disappeared.

For smaller values of the ε parameter—which is the interesting parameter range—the index-based implementations and even Gan and Tao's own GriDBSCAN [31] and KDD'96 implementations again clearly outperform the new algorithm by Gan and Tao [18].

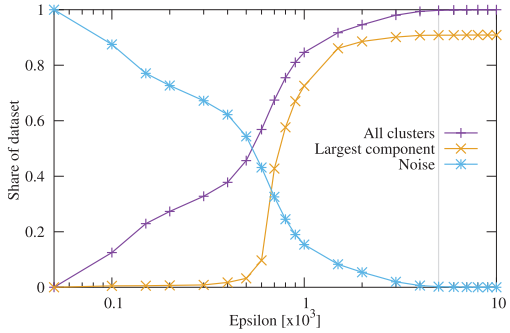
It would also have been good to include other grid-based algorithms derived from DBSCAN in the experiments such as STING [45], and DenClue [24] for evaluation, as well as algorithms



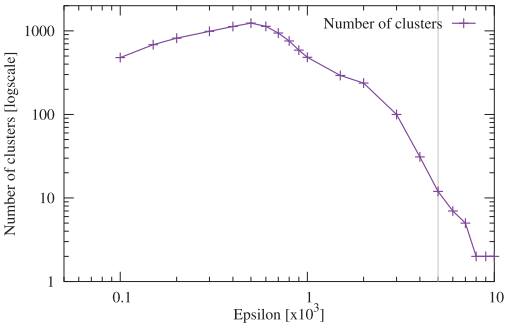
(a) Runtime in log-log-space using the Java implementations from ELKI



(b) Runtime in log-log-space using the implementations by Gan and Tao, and our C++ versions



(c) Cluster sizes in log-log scale



(d) Number of clusters in log-log scale

Fig. 8. Runtime performance on the "PAMAP2" dataset. The vertical line indicates the minimum $\varepsilon = 5{,}000$ used by Gan and Tao.

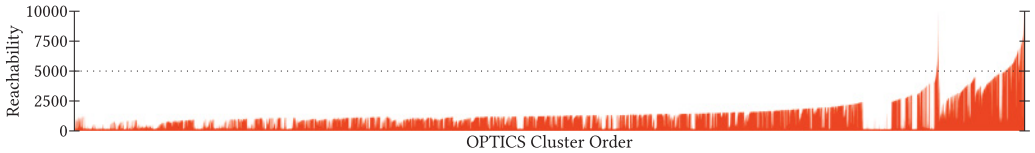
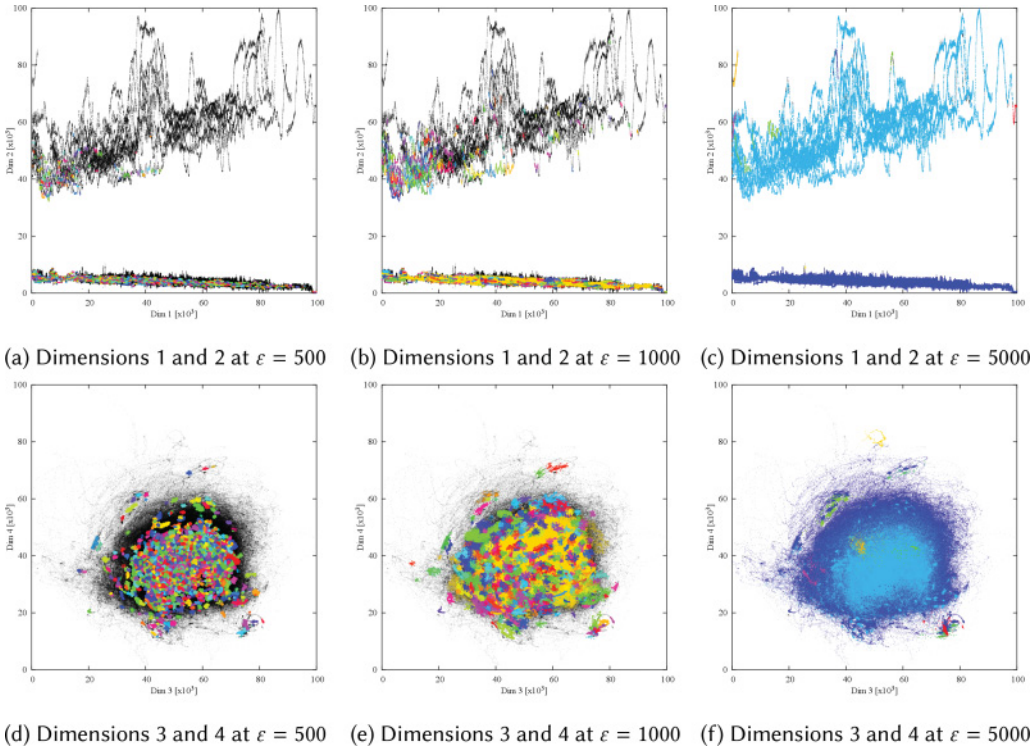


Fig. 9. OPTICS plot for the"PAMAP2" dataset, with a maximum distance of 10,000.

such as OPTICS [3], and HDBSCAN* [13], which remove the need to choose ε. For the proposed approximative algorithm, a comparison to DBSCAN with approximative neighbor search would also have been interesting.

Re-implementation of the algorithms of Gan and Tao is difficult, because the paper is missing crucial details on how to implement it efficiently: the algorithms rely on an efficient implementation of the BCP problem, and the mentioned result by Agarwal et al. [1] does not yield an obvious efficient implementation, but is primarily of interest for its theoretical complexity.

In summary, the values of ε were clearly chosen too high, and the experiments of the SIGMOD 2015 article only demonstrate better performance on questionable parameter settings and datasets not well suited for cluster analysis. With more realistic parameters, the new algorithm is outperformed by index-based algorithms, and even the GriDBSCAN and KDD'96 reimplementations by Gan and Tao.



(a) Dimensions 1 and 2 at $\varepsilon = 500$ (b) Dimensions 1 and 2 at $\varepsilon = 1000$ (c) Dimensions 1 and 2 at $\varepsilon = 5000$

(d) Dimensions 3 and 4 at $\varepsilon = 500$ (e) Dimensions 3 and 4 at $\varepsilon = 1000$ (f) Dimensions 3 and 4 at $\varepsilon = 5000$

Fig. 10. Scatterplot of "PAMAP2," with the resulting clusters at $\varepsilon \in \{500, 1{,}000, 5{,}000\}$.

5 CONCLUSIONS

This article responds to the SIGMOD 2015 article "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation."

In the first part of the article, we point out inaccuracies in the SIGMOD article. While its Theorem 1 shows that a runtime of $O(n \log n)$ cannot be guaranteed, this does not imply, as claimed by Gan and Tao, that "DBSCAN in $d \geq 3$ is computationally intractable in practice even on moderately large n." Rather, DBSCAN remains a method of choice even for large n[6] because many alternatives are in $\Theta(n^2)$ or $\Theta(n^3)$. Furthermore, the theoretical results presented apply only to Euclidean distance, which is a special case of DBSCAN. In the general case of arbitrary non-metric distance measures, the worst case remains $O(n^2 \cdot D)$, and the proposed algorithms of the SIGMOD article are not applicable.

In the second part of the article, we discuss the choice of DBSCAN parameters, in particular the choice of ε. Our analysis shows that the values of ε were clearly chosen too high in the experiments in the SIGMOD article, which means that the reported results are valid only for extreme parameter settings and on datasets not well suited for cluster analysis. Our experiments with the binary and the datasets used in the SIGMOD article demonstrate that the original DBSCAN algorithm with appropriate (i.e., reasonably small ε) parameters and indexes is efficient, with all the indexes providing a 10- to 1,000-fold speedup over linear scan, using implementations of four different

[6]scikit-learn [37] currently recommends only k-means and DBSCAN for "very large `n_samples`" in the clustering documentation: http://scikit-learn.org/0.18/modules/clustering.html.

authors (including the GriDBSCAN and R-tree implementations by Gan and Tao). The SIGMOD implementation of the new algorithm is outperformed both by DBSCAN using spatial indexes, as well as by GriDBSCAN using the SIGMOD authors' own implementations, in a more interesting ε parameter range.[7] To get deeper insights into DBSCAN, it would also be necessary to evaluate with respect to *utility* of the resulting clusters, as our experiments suggest that the datasets used do not yield meaningful clusters. We may thus be benchmarking on the "wrong" datasets (but, of course, an algorithm should perform well on any data).

In summary, the article by Gan and Tao certainly settles the question of a lower bound for the runtime complexity of any algorithm for the Euclidean DBSCAN problem. However, the support for the claim that the proposed method should replace DBSCAN on big data seems inconclusive. We have demonstrated in this technical communication that the original DBSCAN algorithm, together with effective indexes and reasonably chosen parameter values, performs competitively compared to the new method proposed by Gan and Tao.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thorough and constructive work. The authors are also very grateful for the great effort and valuable guidance by the Editor-in-chief Dr. Christian Jensen and Associate Editor Dr. Graham Cormode.

REFERENCES

[1] Pankaj K. Agarwal, Herbert Edelsbrunner, Otfried Schwarzkopf, and Emo Welzl. 1991. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete & Computational Geometry* 6, 1 (1991), 407–422. DOI: http://dx.doi.org/10.1007/BF02574698

[2] Gennady L. Andrienko, Natalia V. Andrienko, Christophe Hurter, Salvatore Rinzivillo, and Stefan Wrobel. 2011. From movement tracks through events to places: Extracting and characterizing significant places from mobility data. In *Proceedings of the 2011 IEEE Symposium on Visual Analytics Science and Technology (VAST)*. 161–170. DOI: http://dx.doi.org/10.1109/VAST.2011.6102454

[3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 49–60. DOI: http://dx.doi.org/10.1145/304182.304187

[4] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. 2004. The priority R-tree: A practically efficient and worst-case optimal R-tree. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 347–358. DOI: http://dx.doi.org/10.1145/1007568.1007608

[5] Sunil Arya and David M. Mount. 1993. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*. 271–280.

[6] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 322–331. DOI: http://dx.doi.org/10.1145/93597.98741

[7] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18, 9 (1975), 509–517. DOI: http://dx.doi.org/10.1145/361002.361007

[8] Stefan Berchtold, Christian Böhm, Bernhard Braunmüller, Daniel A. Keim, and Hans-Peter Kriegel. 1997. Fast parallel similarity search in multimedia databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 1–12. DOI: http://dx.doi.org/10.1145/253260.253263

[9] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT)*. 217–235. DOI: http://dx.doi.org/10.1007/3-540-49257-7_15

[10] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbors. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. 97–104. DOI: http://dx.doi.org/10.1145/1143844.1143857

[7] This does not mean that with additional heuristics, the new method cannot be made to perform more efficiently for smaller values of ε, particularly in lower dimensional spaces. In fact, through personal communication, we are aware that Gan and Tao have developed such an improved version of ρ-DBSCAN, which is presumably faster on small ε values. We look forward to seeing these improvements and supporting experimental results in a future publication.

[11] Ergun Biçici and Deniz Yuret. 2007. Locally scaled density based clustering. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA) 2007*. 739–748. DOI: http://dx.doi.org/10.1007/978-3-540-71618-1_82

[12] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1994. Multi-step processing of spatial joins. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 197–208. DOI: http://dx.doi.org/10.1145/191839.191880

[13] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-based clustering based on hierarchical density estimates. In *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. 160–172. DOI: http://dx.doi.org/10.1007/978-3-642-37456-2_14

[14] Michel Marie Deza and Elena Deza. 2009. *Encyclopedia of Distances (3rd ed.)*. Springer.

[15] Jeff Erickson. 1995. On the relative complexities of some geometric problems. In *Proceedings of the 7th Canadian Conference on Computational Geometry*. 85–90.

[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 *Proceedings of the 2nd ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 226–231.

[17] Christos Faloutsos and Ibrahim Kamel. 1994. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 4–13. DOI: http://dx.doi.org/10.1145/182591.182593

[18] Junhao Gan and Yufei Tao. 2015. DBSCAN revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*. 519–530. DOI: http://dx.doi.org/10.1145/2723372.2737792

[19] Ade Gunawan. 2013. *A faster algorithm for DBSCAN*. Master's thesis. Technical University of Eindhoven. *http://repository.tue.nl/760643*.

[20] Mark A. Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *ACM SIGKDD Explorations* 11, 1 (2009), 10–18. DOI: http://dx.doi.org/10.1145/1656274.1656278

[21] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques (3rd ed.)*. Morgan Kaufmann.

[22] Joseph M. Hellerstein, Elias Koutsoupias, and Christos H. Papadimitriou. 1997. On the analysis of indexing schemes. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 249–256. DOI: http://dx.doi.org/10.1145/263661.263688

[23] Kirsten Hildrum, John Kubiatowicz, Sean Ma, and Satish Rao. 2004. A note on the nearest neighbor in growth-restricted metrics. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 560–561.

[24] Alexander Hinneburg and Daniel A. Keim. 1998. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 4th ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 58–65.

[25] Michael E. Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2010. Can shared-neighbor distances defeat the curse of dimensionality? In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM)*. 482–500. DOI: http://dx.doi.org/10.1007/978-3-642-13818-8_34

[26] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. 604–613. DOI: http://dx.doi.org/10.1145/276698.276876

[27] David R. Karger and Matthias Ruhl. 2002. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*. 741–750. DOI: http://dx.doi.org/10.1145/509907.510013

[28] Robert Krauthgamer and James R. Lee. 2004. Navigating nets: Simple algorithms for proximity search. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 798–807.

[29] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2016. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowledge and Information Systems (KAIS)* (Oct. 2016). DOI: http://dx.doi.org/10.1007/s10115-016-1004-2

[30] Moshe Lichman. 2013. UCI Machine Learning Repository. Retrieved from *http://archive.ics.uci.edu/ml*.

[31] Shaaban Mahran and Khaled Mahar. 2008. Using grid for accelerating density-based clustering. In *Proceedings of 8th IEEE International Conference on Computer and Information Technology (CIT'08)*. 35–40. DOI: http://dx.doi.org/10.1109/CIT.2008.4594646

[32] Son T. Mai, Ira Assent, and Martin Storgaard. 2016. AnyDBC: An efficient anytime density-based clustering algorithm for very large complex datasets. In *Proceedings of the 22st ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 1025–1034. DOI: http://dx.doi.org/10.1145/2939672.2939750

[33] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. 2003. *R-trees have Grown Everywhere*. Technical Report. Retrieved from http://www.rtreeportal.org/.

[34] Jirí Matousek. 1991. Reporting points in halfspaces. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 207–215. DOI: http://dx.doi.org/10.1109/SFCS.1991.185370

[35] Jirí Matousek. 1994. Geometric range searching. *Computing Surveys* 26, 4 (1994), 421–461. DOI: http://dx.doi.org/10.1145/197405.197408

[36] Apostolos Papadopoulos and Yannis Manolopoulos. 1997. Performance of nearest neighbor queries in R-trees. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*. 394–408. DOI: http://dx.doi.org/10.1007/3-540-62222-5_59

[37] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[38] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *Proceedings of the 16th International Symposium on Wearable Computers (ISWC'12)*. 108–109. DOI: http://dx.doi.org/10.1109/ISWC.2012.13

[39] Vasilis Samoladas and Daniel P. Miranker. 1998. A lower bound theorem for indexing schemes and its application to multidimensional range queries. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 44–51. DOI: http://dx.doi.org/10.1145/275487.275493

[40] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. 1998. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery* 2, 2 (1998), 169–194. DOI: http://dx.doi.org/10.1023/A:1009745219419

[41] Erich Schubert, Alexander Koos, Tobias Emrich, Andreas Züfle, Klaus Arthur Schmid, and Arthur Zimek. 2015. A framework for clustering uncertain data. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1976–1979. DOI: http://dx.doi.org/10.14778/2824032.2824115 https://elki-project.github.io/.

[42] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. 2013. Geodetic distance queries on r-trees for indexing geographic data. In *Proceedings of the 13th International Symposium on Spatial and Temporal Databases (SSTD)*. 146–164. DOI: http://dx.doi.org/10.1007/978-3-642-40235-7_9

[43] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. 2006. *Introduction to Data Mining*. Addison Wesley.

[44] R Core Team. 2015. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Retrieved from http://www.r-project.org/.

[45] Wei Wang, Jiong Yang, and Richard R. Muntz. 1997. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)*. 186–195.

[46] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*. 194–205.

[47] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas A. J. Schweiger. 2007. SCAN: A structural clustering algorithm for networks. In *Proceedings of the 13th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 824–833. DOI: http://dx.doi.org/10.1145/1281192.1281280

[48] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. 2012. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining* 5, 5 (2012), 363–387. DOI: http://dx.doi.org/10.1002/sam.11161

Received November 2015; revised March 2017; accepted March 2017