High-Dimensional Density-Based Clustering Using Locality-Sensitive Hashing

Camilla Birch Okkels
IT University of Copenhagen
Copenhagen, Denmark
cabi@itu.dk

Martin Aumüller
IT University of Copenhagen
Copenhagen, Denmark
maau@itu.dk

Viktor Bello Thomsen
IT University of Copenhagen
Copenhagen, Denmark
vibt@itu.dk

Arthur Zimek
IMADA, SDU
Odense, Denmark
zimek@imada.sdu.dk

ABSTRACT

The DBSCAN algorithm is a popular density-based clustering method to find clusters of arbitrary shapes without requiring an initial guess on the number of clusters. While there are methods to run DBSCAN efficiently in low-dimensional data in near-linear time, there remains a need for an efficient DBSCAN algorithm that scales to high-dimensional data. The bottleneck in high-dimensional data is that the range queries necessary in carrying out the algorithm suffer from the curse of dimensionality. In this paper we present the SRRDBSCAN algorithm. This algorithm is an implementation of approximate DBSCAN using locality-sensitive hashing. We prove sub-quadratic running time bounds under reasonable assumptions about the data. An important ingredient in the design of the data structure is the use of a multi-level LSH data structure, which automatically adapts to the density of data points. An extensive empirical analysis shows that the approximation does not significantly impact the quality of the clustering found by the algorithm as compared to the exact DBSCAN clustering. Moreover, our algorithm is competitive with other approaches even in low-dimensional settings, and thus provides a general-purpose DBSCAN implementation for arbitrary data.

1 INTRODUCTION

In recent years the amount of data available has increased dramatically. It is no longer uncommon for data collections to consist of millions if not billions of data points. Furthermore, and particularly driven by deep-learning-based embeddings such as CLIP [42], each data point can have many features (dimensions); for an algorithm to be viable for data analysis tasks, it needs to be able to keep up with the trend of increasing size and dimensions in data.

Clustering data is one of the basic primitives of data mining. There exist many different approaches to clustering. One important approach is k-means and its variants, in which the analyst has a good guess on the number k of clusters (or tries out different values of k). If the number of clusters is not known, or the clusters have a non-convex shape, *density-based approaches* are often the preferred approach. Examples of such methods are DBSCAN [22], HDBSCAN [14, 15], hierarchical clustering [7], and others [13]. In this paper, we focus on DBSCAN.

Informally, the DBSCAN problem asks to group the data in such a way that the clusters themselves should correspond to dense regions of points and should be separated by sparse regions. These regions are created by identifying points with more than a certain threshold of close points at a certain distance threshold. Both the number of points required (`minPts`) and the distance threshold (ε) are parameters set by the analyst. Points satisfying the properties are so called "core points"; they are used to initiate the formation of the dense regions, and based on the notion of ε-reachability, clusters are formed. A formal definition of the problem is given in Section 3.

The simplest way to compute the clustering of n data points requires the calculation of the distances between all pairs of points. In this case the number of required distance computations is $O(n^2)$. For many distance measures, the distance comparison of two points in $\mathbb{R}^d$ can be carried out in time $O(d)$, so the total running time is $O(n^2 d)$. Due to the *curse of dimensionality* [9], approximate techniques are necessary to break this quadratic-time barrier for an arbitrary number of dimensions. Gan and Tao [23] were the first to formally define the c-approximate DBSCAN problem, which relaxes the reachability criterion for clusters: if a pair of core points is $c \cdot \varepsilon$ away, they may belong to the same cluster. The purpose of this paper is to propose efficient implementations of approximate DBSCAN for high-dimensional data based on hashing techniques.

As pointed out by Gan and Tao [23], the running time of the DBSCAN algorithm for arbitrary but constant d was up for dispute for a long time. In their paper, they proved both a lower bound of $\Omega(n^{4/3})$ for exact DBSCAN, and provided a linear time algorithm for an approximate version. While this sounds as good as one can hope for (each point has to be inspected at least once to make a clustering decision for it), the dependency of their algorithm is *exponential* in the dimension of the dataset. As pointed out by Schubert et al. [44], even for small d the suggested implementation did in fact not translate into practical improvements.

The present paper studies the design space of locality-sensitive hashing (LSH [32]) based algorithms to solve the approximate variant of DBSCAN with strong probabilistic guarantees. Our algorithm, to be described in Section 4, provides the following guarantees:

Theorem 1.1 (Informal version of Theorem 4.3). *Let $S \subseteq \mathbb{R}^d$ be the dataset, minPts and ε be the DBSCAN-specific parameters, and $c \geq 1$ be an approximation factor. There exists an algorithm that solves the c-approximate DBSCAN problem under Euclidean*

distance with constant probability. Its expected running time[1] *is*

$$\tilde{O}(minPts^{1-1/c^2} n^{1+1/c^2} d),$$

if the neighborhood of core points is "not too dense."

For example, consider the case that $c = 2$, which means that points at distance ε to 2ε may count for a point being a core point in approximate DBSCAN. If minPts is set to some constant, the expected running time of the algorithm is $O(dn^{1.25})$ *for arbitrary dimensionality*. Even for extreme settings, such as minPts = n, or $c = 1$ (exact DBSCAN), the running time smoothly degrades to guarantee quadratic time in the worst-case.

The backbone of the algorithm is the Multi-level LSH data structure first described by Ahle et al. [4]. On a high level, our algorithm uses this data structure for DBSCAN to reduce the number of distance computations necessary by only considering points hashed together in the data structure. We will both analyze theoretically and observe empirically that different density regions require access to finer granularity of the hash function. In previous hashing-based work such as [36], only a single level is used for all points; this will necessarily translate into quadratic running time for some parameter choices of DBSCAN.

Contributions. The contribution of this paper is the design, implementation, and evaluation of an efficient multi-level LSH data structure that provides a solution to the approximate DBSCAN problem. The proposed algorithm is supported by theoretical guarantees on the time complexity as well as the correctness of the output. The general description of the algorithm of Section 4 uses LSH as a black-box which means that our algorithm can be applied in all distance spaces that are "LSH-able" [18]. Only in Section 5 we describe particular details of the LSH function and other implementation choices that influence empirical performance. The experimental results in Section 6 show competitive performance in running time of the algorithm compared to existing DBSCAN implementations in the literature, even for low-dimensional data. As we will show in the experimental section, none of the competitors that provide theoretical guarantees on the output quality are able to solve both low- and high-dimensional clustering problems on datasets containing millions of data points within reasonable time limits, often failing due to the dimensionality of the datasets, or certain clustering characteristica rendering their approach inefficient—both in terms of running time and memory consumption. Both our implementation, the experimental framework, and additional experimental results are available at https://github.com/CamillaOkkels/srrdbscan and https://github.com/CamillaOkkels/dbscan-benchmark.

We stress that, while LSH is a well-known technique for enabling scalable data analysis, no prior work on (approximate) DBSCAN has both strong theoretical guarantees on the result quality *and* good empirical performance.

2 RELATED WORK

There exist many proposed variations on the original DBSCAN algorithm that try to mitigate its drawbacks. In the 2-dimensional case efficient algorithms exist that have improved the running time from $O(n^2)$ to $O(n \cdot \log(n))$ [20]. Efficient algorithms for the general d-dimensional case as proposed by Gan and Tao [23] as well as Schubert et al. [44]—while achieving sub-quadratic running times—rely on constant dimensionality. Indeed, when the dimensionality becomes high, these algorithms do not gain significant improvement compared to the naïve implementation of DBSCAN. While we focus our efforts on efficient DBSCAN implementations, the same research questions are actively investigated for hierarchical clusterings as well [1, 20, 25, 30, 47].

Subspace clustering. Much research has been spent on clustering in high-dimensional data [28, 39], in particular including some adaptations of DBSCAN such as SUBCLU [35], PreDeCon [10], 4C [11], or COPAC [2]. These and many other methods in this area broadly described as subspace clustering do not, however, focus on efficiency and robustness against the curse of dimensionality when solving the DBSCAN problem as such, but on re-defining the problem as to identify (possibly multiple) clustering solutions in (different) subspaces. These and related methods, although explicitly addressing some issues that come with high dimensional data, namely the different attributions of relevant vs. irrelevant attributes for different clusters, do not address the scalability and robustness issues for the underlying basic DBSCAN problem when it comes to very large and high-dimensional data.

Efficient implementations. Most approaches to speeding up DBSCAN rely on using an indexing structure to allow for faster (range) search operations. These structures usually split up the high-dimensional space into a grid; since the number of grid cells increases exponentially in the dimensionality d, such approaches will often have depend exponentially on the dimensionality. An efficient implementation of DBSCAN was proposed by Gan and Tao in [23, 24] for 2-dimensional as well as d-dimensional DBSCAN. The 2D algorithm solves DBSCAN in time $O(n \cdot \log(n))$ by imposing a grid and using the cells to reduce the number of distance computations for core point identification. They presented an approximate and exact algorithm for solving DBSCAN for $d \geq 3$. The data structure for the approximate algorithm is a quadtree-like hierarchical grid structure. They achieve a running time of $O(n)$ for constant d, where the running time depends exponentially on d. The state-of-the-art grid-based approach is TPEDBSCAN by Wang et al. [46] which focused on an efficient parallel implementation of the grid-based approach; it has the same exponential dependency on d.

Another line of work based the computation of DBSCAN clusterings on sampling approaches. SNGDBSCAN by Jiang et al. [33] is the state-of-the-art approach in this regime. For constant dimensionality d (again hiding an exponential dependency), they prove that sampling $O(n \log n)$ random pairs of points is sufficient to identify the DBSCAN clusters if "clusters are sufficiently dense."

Hashing-based clustering. Koga et al. [37] and Keramatian et al. [36] both rely on LSH to design clustering algorithms that work for high-dimensional data. In contrast to grid-based approaches that impose a grid structure, the main idea of these hashing-based algorithms is to hash data points into (multiple) hash tables. Traditionally, this is achieved using LSH. The promise of using LSH is that the probability of hashing to the same hash value is a function increasing in the similarity of the data points. The main bottleneck of all hashing-based algorithms is that in each bucket of a hash table, an all-to-all comparison has to be carried out to find close neighbors for all data points in the bucket, yielding *quadratic time in the bucket size*. Thus, care has to be taken in choosing parameters such that (i) buckets are not too large and (ii) enough repetitions are carried out to guarantee core point identification with high enough probability. Koga et al. [37] introduce an algorithm called *LSH-LINK* to compute a hierarchical clustering. However, their running time guarantees might exceed $O(n^2)$, and—more importantly—there is no correctness

[1]With the $\tilde{O}(.)$ notation we suppress polylogarithmic factors in n and *minPts*.

guarantee for their algorithm. Keramatian et al. [36] recently provided an efficient clustering algorithm called *IP.LSH.DBSCAN* (IP.LSH) that computes a single reference point per bucket. This heuristic alleviates the quadratic-time barrier in each bucket, but does not provide strong formal guarantees [36, Lemma 2], which we will also observe in the experimental evaluation (Section 6.2). They also require additional parameters to set the number of hash functions and hash tables to which the algorithm is very sensitive, also to be explored in the experimental section. Other LSH-based approaches with no theoretical guarantees on the result quality include [45, 48].

Other approximate methods. While not being hashing-based, random projection-based ideas have been explored to cluster data. This idea was pioneered by Schneider et al. [43]. In very recent work, Xu and Pham [49] designed an efficient algorithm for DBSCAN using random projections. While they give theoretical guarantees on certain subroutines, there is no general running time statement and the correctness of the merging step requires the same strong assumptions [49, Assumption 1] as sampling-based approaches [33].

3 PRELIMINARIES

3.1 Definitions

In the following, assume a data set $S \subseteq \mathbb{R}^d$ consisting of n points is given. Let $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be some distance measure, for example the Euclidean distance $d(p, q) = \|p - q\|_2$. For a point $q \in \mathbb{R}^d$ and a distance threshold $\varepsilon \geq 0$, define $N_\varepsilon(q) = \{p \in S \mid d(p, q) \leq \varepsilon\}$ as the ε-neighborhood of q in S. While certain core concepts of DBSCAN [22] are standard, in particular the role of border points in clusters is often vaguely discussed. We consider the DBSCAN* problem [15] in which border points are not part of a clustering and define an approximate version as follows:

Definition 3.1 (c-approximate core point sets). Let $q \in S$ be a data point, let ε and *minPts* be the DBSCAN parameters, and let $c \geq 1$ be an approximation factor. If $|N_\varepsilon(q)| \geq$ *minPts*, q is a *core point*. If $|N_\varepsilon(q)| <$ *minPts* and $|N_{c\varepsilon}(q)| \geq$ *minPts*, q is a *c-approximate core point*. Let CP^* be the set of all core points in S. A set $CP \subseteq S$ is a *c-approximate core point set* if $CP^* \subseteq CP$ and all points in $CP \setminus CP^*$ are c-approximate core points.

We remark that for $c = 1$, the core point set is the unique set of core points in S in the classical DBSCAN notion. For $c > 1$, we allow that the core point set moreover includes points that are only core points at distance threshold $c\varepsilon$, but we do not require that all of these are included in the c-approximate core point set.

Definition 3.2 (c-approximate density-reachability). Let CP be an c-approximate core point set. A point p is said to be density reachable in CP from another point q if there exists a sequence of points $(p := p_1, p_2, ..., p_\tau =: q)$ from CP (with $\tau \geq 2$) that satisfies $d(p_i, p_{i+1}) \leq \varepsilon$ for each $i \in [1, \tau - 1]$.

Definition 3.3 (c-approximate DBSCAN clustering).* Given c, ε, and *minPts*, let $CP \subseteq S$ be an c-approximate core point set. Let $\sim_{CP}$ be the relation such that $p \sim_{CP} q$ if and only if p and q are c-approximately density-reachable in CP. The $(c, \varepsilon,$ *minPts*$)$-approximate DBSCAN* clustering $\mathscr{C}$ of CP are the equivalence classes of $\sim_{CP}$. The remaining points $\{p \in S \setminus CP\}$ are noise.

Alternatively, we can see the clusters as the connected components induced by the c-approximate core point set CP of the graph $G = (S, E)$, where an edge exists if two points are at distance at most ε [20]. For $c = 1$, we recover the exact DBSCAN* clustering.

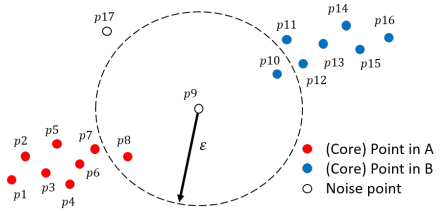


Figure 1: An example of a DBSCAN* clustering with the depicted ε and *minPts* = 4. The clusters only consist of core points.

Figure 1 shows a DBSCAN* clustering according to Definition 3.3. The clusters returned by DBSCAN* are $\{p1, p2, ..., p8\}$ and $\{p10, ..., p16\}$. Both $p9$ and $p17$ are ignored because they are not a core point.

Both [24] and [20] use a different notion of approximation. Their core point classification is exact, but Definition 3.2 (iii) uses $c \cdot \varepsilon$ as a distance value. While the individual approximate clustering might differ, it is straight-forward to see that the quality of the resulting approximate clustering can be "sandwiched" in between two exact solutions. Thus, these different notions are similar in their quality.

Theorem 3.4 (Sandwich theorem [24]). *Given $S \subseteq \mathbb{R}^d$, ε, minPts, and $c \geq 1$.*

- *Let $\mathscr{C}_\varepsilon$ be the clusters for $(\varepsilon,$ minPts$)$-DBSCAN*.*
- *Let $\mathscr{C}_c$ be a $(c, \varepsilon,$ minPts$)$-approximate DBSCAN* clustering.*
- *Let $\mathscr{C}_{c\cdot\varepsilon}$ be the clusters for $(c\varepsilon,$ minPts$)$-DBSCAN*.*

Then we have: (1) For each $C_\varepsilon \in \mathscr{C}_\varepsilon$ there exists $C_c \in \mathscr{C}_c$ such that $C_\varepsilon \subseteq C_c$. (2) For each $C_c \in \mathscr{C}_c$ there exists $C_{c\cdot\varepsilon} \in \mathscr{C}_{c\cdot\varepsilon}$ such that $C_c \subseteq C_{c\cdot\varepsilon}$.

Theorem 3.4 is a natural consequence of the well documented fact (see for example [7, 25]) that exact DBSCAN* clusters for small epsilon are subsets of DBSCAN* clusters for larger epsilon. In particular, all core points of $(\varepsilon,$ *minPts*$)$-DBSCAN* are also c-approximate core points, and all non-core points in $(c\varepsilon,$ *minPts*$)$-DBSCAN* are also non-core points in the approximate setting. The strength of this theorem lies in the fact that an approximate DBSCAN* definition can be sandwiched between two exact solutions. Consider the instances where the exact DBSCAN* clusters are robust up to a small variation in ε, i.e., $\mathscr{C}_\varepsilon = \mathscr{C}_{c\cdot\varepsilon}$. Theorem 3.4 states that the solution to approximate DBSCAN* with parameters $(c, \varepsilon,$ *minPts*$)$ would also be the solution to exact DBSCAN*. In other words the similarity between the solution to exact DBSCAN* and approximate DBSCAN is entirely dependent on the robustness of the clusters.

3.2 Objective of DBSCAN* algorithms

The objective of DBSCAN* is to find a clustering $\mathscr{C}$ given parameters ε and *minPts* that satisfy Definition 3.3 with $c = 1$. Any algorithm for solving the DBSCAN* problem should have the following properties: (i) It needs to identify core points given the parameters ε and *minPts*. (ii) Given the list of core points CP and ε, clusters have to be identified based on density-reachability.

Algorithm 1: SRRDBSCAN($S, \varepsilon, minPts, L, \delta, c$)

Input: $S \subseteq \mathbb{R}^d$, parameters ε, $minPts$, L, δ, c
Output: A list of cluster labels.

1 $\mathcal{I} \leftarrow$ build multi-level LSH for S, L, δ // Sect. 4.1
2 $k_{\text{best}} \leftarrow$ findBestLevel($S, \mathcal{I}, L$) // Sect. 4.3
3 $CP \leftarrow$ classifyCP($S, \mathcal{I}, k_{\text{best}}, \varepsilon, minPts, c$) // Sect. 4.3
4 clusters $\leftarrow$ clusterCore(CP, ε, L, δ) // Sect. 4.4
5 **return** *clusters*

3.3 Locality-Sensitive Hashing

To reduce distance computations we use Locality-Sensitive Hashing [32]. Let $\Pr(\chi)$ denote the probability of an event χ.

Definition 3.5 (Locality-Sensitive Hash Family [32]). Given a distance function d, a family of hash functions $\mathcal{H} = \{h : \mathbb{R}^d \to R\}$ is said to be (d_1, d_2, p_1, p_2)-sensitive if for any pair of points $p, q \in \mathbb{R}^d$ the following statements are true:

(i) For $d(q,p) \le d_1$, $\Pr_{h\sim\mathcal{H}}(h(q) = h(p)) \ge p_1$.
(ii) For $d(q,p) \ge d_2$, $\Pr_{h\sim\mathcal{H}}(h(q) = h(p)) \le p_2$.
(iii) $p_1 \ge p_2$ and $d_1 \le d_2$.

Many LSH families have been described in the past. Of particular importance are BitSampling [32] for binary strings under Hamming distance, MinHash [12] for sets, SimHash [17] and Crosspolytope LSH [6] for angular distance on the unit sphere, and EuclideanLSH [19] for Euclidean space. As it is standard in the literature [3], we assume that p_1 is constant. The quality of an LSH is measured as $\rho = \log(1/p_1) / \log(1/p_2) \le 1$.

4 PROPOSED ALGORITHM

In this section, we first describe our algorithm for solving the approximate DBSCAN* problem. Algorithm 1 outlines the algorithm SRRDBSCAN (Spherical Range Reporting DBSCAN). In addition to the DBSCAN* parameters S, ε, and $minPts$, and the approximation factor c, our algorithms takes two user parameters $\delta \in (0,1)$ (which controls the success probability) and $L \ge 1$ (which governs the space consumption of the index data structure). In Sections 4.1—4.4 we discuss the proposed algorithm in detail, including the index building phase, the core point identification, and merging core points into the final clustering. In Section 4.5 we argue for the algorithm's correctness and then analyze its running time.

4.1 Index building

The proposed algorithm makes use of the Multi-Level LSH data structure first described by Ahle et al. [4]. Given a dataset $S \subseteq \mathbb{R}^d$, a parameter L, and access to a (p_1, p_2, d_1, d_2)-sensitive hash family $\mathcal{H}$ consisting of function that map from $\mathbb{R}^d$ to some range R, it builds a multi-level data structure as follows. For a level $k, 0 \le k \le K$, in the data structure, choose k hash functions $\{g_1, g_2, ..., g_k\}$ uniformly at random and independently from $\mathcal{H}$ to obtain the hash function $h_k = (g_1, g_2, ..., g_k)$. The hash value of a point $x \in \mathbb{R}^d$ is $h_k(x) = (g_1(x), \ldots, g_k(x)) \in R^k$. For each $x \in S$, compute $h_k(x)$ and, using a hash table, associate with each hash value the list of points that hash to it. Repeat this process $\text{reps}(k)$ many times independently, where $\text{reps}(k)$ is a parameter to be set later. The data structure for level k consists of these $\text{reps}(k)$ hash tables $T_{k,1}, \ldots, T_{k,\text{reps}(k)}$. We denote by $T_{k,i}(p)$ the bucket that point p was hashed to on level k and repetition i. Repeat this process independently for each $k \in \{0, \ldots, K\}$, where K is defined to be the largest integer satisfying $\text{reps}(K) \le L$. Figure 2 provides an example. For reasons that will be addressed in the analysis, L has to be large enough such that $K \ge \left\lceil \frac{\log(n/minPts)}{\log(1/p_2)} \right\rceil$.

4.2 Motivation for the data structure

Assume some index $T_{k,j}$ for $k \in \{0, \ldots, K\}$ and $1 \le j \le \text{reps}(k)$ built for a dataset S. Let ε, minPts be the DBSCAN parameters, and let c be the approximation factor. For each $x \in S$ we have to identify if it is a core point and (potentially) merge cluster labels. Let us first consider the influence of the different levels $k \in \{0, \ldots, K\}$. For a given level k, concatenating k hash functions chosen independently at random from $\mathcal{H}$ results in higher selectivity of the hash function: if two points are at distance at most $d_1 := \varepsilon$, their collision probability will be *at least* p_1^k, if they are at least $d_2 := c\varepsilon$ far apart, it is *at most* p_2^k (Def. 3.5). Let $p \in S$ be an arbitrary point. Given the properties of the LSH family, at most $n \cdot p_2^k$ points that are not within distance $c\varepsilon$ are going to collide with p under the hash function. However, a point that counts towards p being a core point has probability at least p_1^k of colliding with p. Repeating the construction $\Theta(1/p_1^k)$ times guarantees that it collides with p with constant probability. Thus, we can *upper bound* the work for deciding whether point p is a core point on level k by $O(p_1^{-k} \cdot (minPts + np_2^k))$. Yet, the actual work depends heavily on the data: If the dataset consists of clusters, each of size $minPts + 1$ with very small intra-cluster distances, but large inter-cluster distances, then a small level k suffices to split up the clusters and the upper bound from before would clearly misguide us to use a larger level with higher cost. In the most extreme case where we have n copies of the same data point, we would spend time $O(minPts \cdot n/p_1^k)$, so the best level is the first. But if the dataset contains a long "chain" of core points at distance ε to each other, and the remaining dataset is noise, slightly further away than distance $c \cdot \varepsilon$, then we would indeed need to inspect a deeper level in which these noise points are likely to be split up from core points. Hence we propose to first compute the work that needs to be carried out on level k (which is easier since we only have to sum up the number of colliding points, i.e., the bucket sizes in the hash table), choose the level with minimal cost to carry out the core point identification, and later merge clusters. The multi-level data structure allows for *data dependent* choices.

4.3 Core point identification

To identify c-approximate core points, i.e., the points that have at least $minPts$ many neighbors at distance ε (where we might count points at distance at most $c\varepsilon$), we proceed in two steps. First, using Algorithm 2, we determine a best level to carry out the core point identification in the multi-level data structure. To do so, we compute in Line 3 the amount of work that will be necessary to inspect all buckets on level k and to carry out all-to-all comparisons in each of them (without actually carrying them out). We proceed in the order $k = 0, 1, 2, \ldots$ and stop as soon as the time necessary for inspecting the buckets is already more than carrying out these all-to-all comparisons in the best level found so far. Let k_{best} be the level returned by Algorithm 2. Algorithm 3 describes the core point identification given such a best level. For a point $p \in S$, it counts the number of distinct points in the buckets $T_{k_{\text{best}},1}(p), \ldots, T_{k_{\text{best}},\text{reps}(k_{\text{best}})}(p)$ that are at distance at most $c \cdot \varepsilon$. It stops as soon as the number is $minPts$, and classifies the point as a core point.

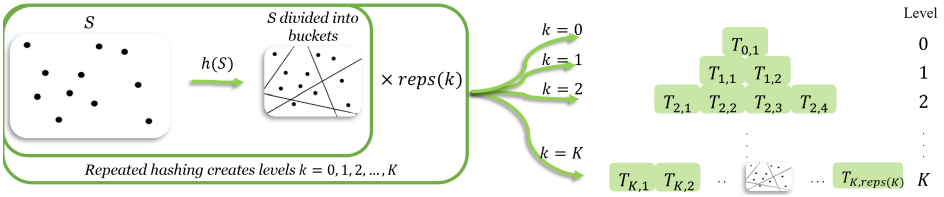


Figure 2: An illustration of the multi-level data structure. Given dataset S, starting from $k = 0$, the data is hashed $\text{reps}(k)$ times for each level k to be constructed. In this example, $\text{reps}(k) = 2^k$. This is repeated until the final level K for which $\text{reps}(K) \leq L$.

Algorithm 2: findBestLevel($S, \mathcal{I}, L$)

Input: Points S, data structure $\mathcal{I}$, and a parameter L
Output: Value k_{best} of "best level"

$n \leftarrow |S|, \quad k \leftarrow 0, \quad k_{\text{best}} \leftarrow 0, \quad \omega_{k_{\text{best}}} \leftarrow n^2$
while $n \cdot reps(k) \leq \min(n \cdot L, \omega_{k_{best}})$ **do**
  $\omega_k \leftarrow \sum_{q \in S} \sum_{i=1}^{\text{reps}(k)} \frac{|T_{k,i}(q)|-1}{2} + 1$
  if $\omega_k \leq \omega_{k_{best}}$ **then**
    $k_{\text{best}} \leftarrow k, \quad \omega_{k_{\text{best}}} \leftarrow \omega_k$
  $k \leftarrow k + 1$
return k_{best}

Algorithm 3: classifyCP($S, \mathcal{I}, k_{\text{best}}, \varepsilon, minPts, c$)

Input: Points S, data structure $\mathcal{I}$, a best level k_{best}, and parameters ε, $minPts$, c
Output: List CP of core points

$CP \leftarrow \{\}$
foreach $p \in S$ **do**
  $N(p) = \{\}$
  for $i \leftarrow 1$ **to** $reps(k_{best})$ **do**
    foreach $q \in T_{k_{best},i}(p)$ **do**
      if $d(p, q) \leq c \cdot \varepsilon$ ***and*** $q \notin N(p)$ **then**
        $N(p) \leftarrow N(p) \cup \{q\}$
      if $|N(p)| = minPts$ **then**
        $CP \leftarrow CP \cup \{p\}$
        break (out of line 4 loop)
return CP

4.4 Merging core points

Algorithm 4 shows how to identify clusters given the list CP of core points identified in the previous step. First, initialize a union-find data structure. Given CP, rebuild the multi-level data structure on CP and identify the best level k_{best} using Algorithm 2. Next, identify *all* the points at distance at most ε from a core point p by going through buckets $T_{k_{\text{best}},1}(p), \ldots, T_{k_{\text{best}},\text{reps}(k_{\text{best}})}(p)$. For each point q found in the buckets, check if the distance is at most ε and merge the cluster using a union operation if that is the case. The clustering $\mathscr{C}$ is the sets of the union-find data structure.

4.5 Analysis of the algorithm

4.5.1 Correctness guarantee. The following lemma says that the algorithm works correctly with probability at least $1 - \delta$ no matter which level is chosen for the individual points in Algorithm 3.

Lemma 4.1 (Correctness guarantee). *Let $S \subseteq \mathbb{R}^d$ contain n data points. Let CP^* be the set of core points with respect to $(\varepsilon, minPts)$-DBSCAN* on S. Set*

$$reps(k) = p_1^{-k} \cdot \ln\left(n \cdot minPts \cdot (k+1)^2/\delta\right)$$

and let CP be the output of Algorithm 3. With probability at least $1-\delta$, $CP^* \subseteq CP$ and all points in CP are c-approximate core points.

For the proof, we will make use of the following lemma:

Lemma 4.2. *Let $p \in S$ be a core point with respect to $(\varepsilon, minPts)$-DBSCAN* and given some $\delta \in (0, 1)$. Then for*

$$reps(k) \geq p_1^{-k} \cdot \ln(minPts/\delta),$$

p is identified as a core point by Alg. 3 with probability at least $1 - \delta$.

Algorithm 4: clusterCore($CP, \varepsilon, L, \delta$)

Input: A list CP of core points and parameters ε, L, δ
Output: A list of cluster labels/identifiers for each point.

$uf \leftarrow$ UnionFind($|CP|$) ; `// Initialize Union-Find data structure to track clusters`
$\mathcal{I} \leftarrow$ build multi-level LSH for CP, L, δ;
$k_{\text{best}} \leftarrow$ findBestLevel($CP, \mathcal{I}, L$)
foreach $p \in CP$ **do**
  for $i \leftarrow 1$ **to** $reps(k_{best})$ **do**
    foreach $q \in T_{k_{best},i}(p)$ **do**
      if uf.`find`$(p) \neq uf$.`find`(q) ***and*** $d(p, q) \leq \varepsilon$ **then**
        uf.`union`(p, q);
return $\{uf$.`find`$(p) \mid p \in CP\}$;

Proof. Fix a level $k \in \{0, \ldots, K\}$. Let p be a core point in S. Given a point q with $d(q, p) \leq \varepsilon$, let A_q be the event that p and q do not get hashed together in any hash table on level k. Since all repetitions are independent and by the definition of LSH, $\Pr(A_i) \leq (1 - p_1^k)^{\text{reps}(k)}$. Now, w.l.o.g., suppose $|N_\varepsilon(p)| = minPts$.[2] In order to identify p as a core point, it should be hashed with every point in $N_\varepsilon(p)$ in at least one hash table on the level. Using a union bound we can find an upper bound on the probability that there exists a hash table where p is not hashed with some $q \in N_\varepsilon(p)$:

$$\Pr\left(\bigcup_{q \in N_\varepsilon(p)} A_q\right) \leq \sum_{j=1}^{minPts} \left(1 - p_1^k\right)^{\text{reps}(k)} = minPts \cdot \left(1 - p_1^k\right)^{\text{reps}(k)}$$

[2] If $|N_\varepsilon(p)| \geq minPts$, the probability that p gets hashed with $minPts$ close points, thereby identifying it as a core point, is only higher.

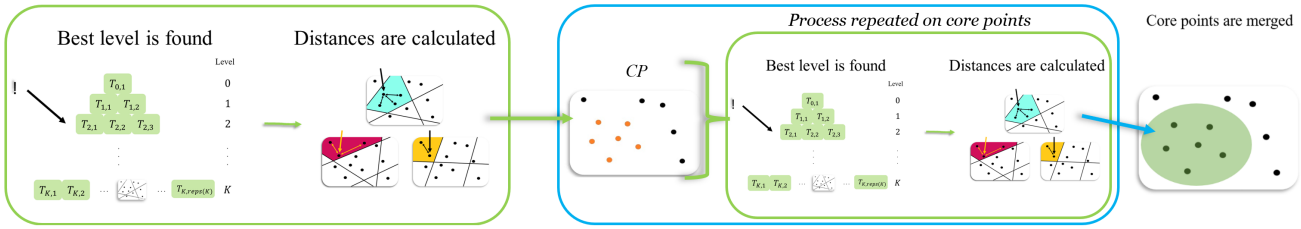


Figure 3: Illustration of the SRRDBSCAN algorithm. Part (a) illustrates the algorithm up to and including core point identification outlined in lines 1-3 in Algorithm 1. Part (b) illustrates the merging step outlined in Algorithm 4. The process of building the data structure and searching for close points is repeated, but only on core points, and the output is clusters.

Using the upper bound $(1-p_1^k)^{\text{reps}(k)} \leq e^{-p_1^k \cdot \text{reps}(k)}$ we get:

$$\Pr\left(\bigcup_{i=1}^{minPts} A_i\right) \leq minPts \cdot e^{-p_1^k \cdot \text{reps}(k)} \tag{4.1}$$

Solving $minPts \cdot e^{-p_1^k \cdot \text{reps}(k)} \leq \delta$ for $\text{reps}(k)$ ends the proof. □

Proof of Lemma 4.1. Let B_i be the event for any given core point i that it is not identified at a level k. From (4.1) we have that $\Pr(B_i) \leq minPts \cdot e^{-p_1^k \cdot \text{reps}(k)}$. Using a union bound over all n potential core points, we can upper bound the probability that at least one core point is not identified on at least one level as

$$\Pr\left(\bigcup_{i=1}^{n\cdot(K+1)} B_i\right) \leq \sum_{k=0}^{K} n \cdot minPts \cdot e^{-p_1^k \cdot \text{reps}(k)}$$

By setting $\text{reps}(k) \geq p_1^{-k} \ln\left(2 \cdot n \cdot minPts \cdot (k+1)^2/\delta\right)$, we may bound this probability by:

$$\Pr\left(\bigcup_{i=1}^{n\cdot(K+1)} B_i\right) \leq \sum_{k=0}^{K} \frac{\delta}{2\cdot(k+1)^2} \leq \delta \cdot \frac{\pi^2}{12} < \delta \ ,$$

where the last inequality uses the fact that $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$. □

4.5.2 Running time analysis. We will prove the following theorem summarizing the expected running time of the proposed algorithm:

Theorem 4.3 (Running time). *Let $S \subseteq \mathbb{R}^d$ with n points be given. Let $minPts, \varepsilon$ be the DBSCAN parameters, and let $L, c \geq 1, \delta > 0$ be user parameters. Define $\mathcal{N} = \sum_{q\in S} |N_{c\cdot\varepsilon}(q)|^{\rho(c)}$. With probability at least $1-\delta$, the output of Algorithm 1 is a $(c, minPts, \varepsilon)$-approximate DBSCAN* clustering. Assuming that hash function evaluations and distance comparisons take time $O(d)$, the expected running time is*

$$\tilde{O}\Big(d \cdot \max\left(nKL, n^{1+\rho(c)}\, minPts^{1-\rho(c)}, n^{\rho(c)}\, minPts^{-\rho(c)}\, \mathcal{N}\right)\Big).$$

The value $\rho = \log(1/p_1)/\log(1/p_2) \leq 1$ is governed by the LSH family that is used for a given distance measure. For example, EuclideanLSH [19] has $\rho(c) = 1/c$, and there exist families that achieve $1/c^2$ [5]. The three terms in Theorem 4.3 represent the time to build the index, the expected time to identify all core points, and the expected time necessary to merge core points into clusters. If $|N_{c\varepsilon}(q)| = O(minPts)$, i.e., if the number of points at distance at most $c\varepsilon$ is not much larger than $minPts$, then the term $n^{\rho(c)}\, minPts^{-\rho(c)}\, \mathcal{N}$ is upper bounded by $O\left(n^{1+\rho(c)}\, minPts^{1-\rho(c)}\right)$. This means that the running time of the merging step is no higher than the time spent in the core point identification step.

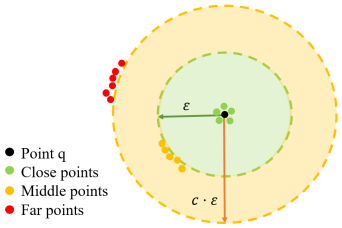


Figure 4: An illustration of the three different categories of points as seen from the perspective of a point q.

In the following, we analyse each of the three steps separately, only counting the number of points inspected by the algorithm (i.e., disregarding the $O(d)$ term present in Theorem 4.3).

Index building. From Lemma 4.2, we build the data structure for all levels $k \in \{0, \ldots, K\}$ such that $\text{reps}(K) \leq L$, using that $\text{reps}(k) = p_1^{-k} \cdot \ln\left(n \cdot minPts \cdot k^2/\delta\right)$. There are $O(L)$ hash tables in the multi-level data structure, as $\sum_{0\leq k\leq K} \text{reps}(k) = O(\text{reps}(K))$. For each hash table, we have to evaluate not more than nK hash values and insert n points into the hash table. Thus, the overall time necessary to build the index is $O(nKL)$.

Core point identification. We proceed in two steps: First, we show that a good level exists in the data structure; next, we rely on the properties of the Multi-Level LSH data structure proven in [4] to argue that the adaptive level choice does not lead to a worse asymptotical running time.

Lemma 4.4. *Set $k = \left\lceil \frac{\log(n/minPts)}{\log(1/p2)} \right\rceil$. Carrying out Algorithm 3 on level k, the expected number of distance comparisons (Line 6) is $\tilde{O}\left(n^{1+\rho}\, minPts^{1-\rho}\right)$.*

Proof. By linearity of expectation, the expected number of distance comparisons over all n data points is just n times the expected number of distance comparisons for a single point.

Fix a point $q \in S$. There are three types of points it can get hashed with into the same bucket in $T_{k,i}$:

(1) Close points (CL): All the points p with $d(q,p) \leq \varepsilon$. These count towards q being a core point. This happens with probability at most 1.
(2) Middle points (MP): All the points p with $\varepsilon \leq d(q,p) \leq c\cdot\varepsilon$. These count towards q being a core point. This happens with probability at most p_1^k.
(3) Far points (FP): All the points p with $d(q,p) \geq c \cdot \varepsilon$. This happens with probability at most p_2^k.

Figure 4 visualizes a setting in which these probabilities are attained, i.e., a worst-case setting for the analysis. The expected

size of the bucket $T_{k,i}(q)$ is given by $\sum_{p \in S} \Pr(h(q) = h(p))$ and can be upper bounded by $E\left[\left|T_{k,i}(q)\right|\right] \leq \#CL + \#MP \cdot p_1^k + \#FP \cdot p_2^k$.

The first two summands of the summation of the right-hand side are at most *minPts*, since Algorithm 3 immediately returns as soon as *minPts* neighbors at distance at most $c\varepsilon$ are identified. By the choice of k, we may bound the final summand by $\#FP \cdot p_2^k \leq n \cdot p_2^k \leq minPts$. So, $E\left[\left|T_{k,i}(q)\right|\right] \leq minPts$ for a single repetition i. The excepted number of distance computations for q is then $O(\text{reps}(k) minPts)$ with $\text{reps}(k) = \tilde{O}(p_1^{-k})$ (cf. Lemma 4.1), which is

$$\tilde{O}\big((n/minPts)^{\rho} minPts\big) = \tilde{O}\big(n^{\rho} minPts^{1-\rho}\big)$$

. The proof follows from linearity of expectation over the n data points. □

While Lemma 4.4 proves that there exists a good level to check, the proposed algorithm first investigates the bucket sizes to make an adaptive choice for which level is the best to use. A core contribution of Ahle et al. [4] states that this adaptive choice has asymptotically (almost) the same running time as if the best level had been known. To extend their Theorem 3 to our setting, first define

$$E(W_k) = \sum_{q \in S} \sum_{1 \leq i \leq \text{reps}(k)} \Big(O(1) + \sum_{x \in T_{k,i}(q)} \Pr\big(h(q) = h(x)\big)\Big) \quad (4.2)$$

$$W^* = \min_{0 \leq k \leq K} E(W_k)$$

This—purely abstractly—represents the expected work on the optimal level if the whole data distribution were known.

Lemma 4.5. *Let $S, \varepsilon, minPts, \delta$ be given. Let W^* be defined as above. The expected running time of Algorithm 2 and Algorithm 3 is $O(W^*)$.*

Proof. The proof will follow closely that of Ahle et al. [4, Theorem 3]. First, consider Algorithm 2.

Given constant access to the size of the buckets, Line 3 takes time $O(n \cdot \text{reps}(k))$. Suppose the last value of k before the loop starting in Line 2 terminates is k^*. Using the choice of $\text{reps}(k)$ from Lemma 4.1, all iterations of the loop until that point in time took expected time

$$\begin{aligned}\sum_{k=1}^{k^*} & O\big(n \cdot \ln\left(n \cdot minPts \cdot k^2/\delta\right) \cdot p_1^{-k}\big) \\ & \leq n \cdot \ln\left(n \cdot minPts \cdot (k^*)^2/\delta\right) \cdot p_1^{-k^*} \sum_{k=0}^{\infty} O\big(p_1^k\big) \\ & \leq O(\omega_{k_{\text{best}}}).\end{aligned}$$

The last step follows from the while condition which ensures that $n \cdot \text{reps}(k^*) \leq \omega_{k_{\text{best}}}$.

Algorithm 3 looks at the buckets of k_{best} and computes exactly $\omega_{k_{\text{best}}}$ distances. By Jensen's inequality,

$$E[\omega_{k_{\text{best}}}] = E[\min_{0 \leq k \leq K}(\omega_k)] \leq \min_{0 \leq k \leq K} E[\omega_k]$$

which equals $\min_{0 \leq k \leq K} E[W_k] = W^*$. □

We note that this result is slightly different from the result of Ahle et al. [4, Theorem 1], which observed a slight asymptotic difference between the "clairvoyant" algorithm that knows the best level and the adaptive variant. This is because our definition of $E[W_k]$ in (4.2) uses the same repetition count as the adaptive algorithm. In the setting of [4], they have an asymptotic blow-up because they carry out a different number of repetitions. Lemmas 4.4 and 4.5 together show that the expected running time of Algorithms 2 and 3 is $\tilde{O}(n^{1+\rho} minPts^{1-\rho})$, and can potentially be much better.

Merging clusters. We analyze the running time of Algorithm 4.

Lemma 4.6 (Running time of merging step). *Given a set CP of c-approximate core points, let $\mathcal{N} = \sum_{q \in CP} |N_{c \cdot \varepsilon}(q)|$. Algorithm 4 labels the core points in expected time*

$$\tilde{O}(|CP|^{\rho(c)} \cdot minPts^{-\rho(c)} \cdot \mathcal{N}).$$

The proof follows the same steps as seen for the proofs of Lemma 4.4 with the main difference being that Algorithm 4 cannot return after finding just *minPts* close points as we observe the merging of two sub-clusters can depend on a single core point. As a result the expected bucket sizes—and consequently the running time—will depend on the neighborhood sizes.

Proof. Let *CP* be the set of core points. We observe that for any core point $|N_{c \cdot \varepsilon}(q)| \geq minPts$. This means $|CP| \cdot p_2^k \leq |N_{c \cdot \varepsilon}(q)|$. As in the proof of Lemma 4.4, the expected contribution from a bucket is given by:

$$E\left[\left|T_{k,i}(q)\right|\right] \leq \#CL + \#MP \cdot p_1^k + \#FP \cdot p_2^k \leq 2 \cdot |N_{c \cdot \varepsilon}(q)|. \quad (4.3)$$

Let k_{best} be the best level found by the algorithm. The expected time cost T to carry out the for loop (Line 4 in Algorithm 4) is

$$T \leq 2 \cdot \text{reps}(k_{\text{best}}) \cdot \sum_{q \in CP} |N_{c \cdot \varepsilon}(q)|. \quad (4.4)$$

Substituting *minPts* in Lemma 4.2 by $|N_{c \cdot \varepsilon}(q)|$ in the proof of Lemma 4.2, and observing that for any core point $|N_{c \cdot \varepsilon}(q)| \leq |CP|$, we conclude that setting $\text{reps}(k) = p_1^{-k} \cdot \ln\left(\frac{|CP|^2 \cdot k^2}{\delta}\right)$ ensures all pairs of core points q, p for which $d(q,p) \leq \varepsilon$ are hashed together at least once. Similar to the core points identification step, to ensure $|CP| \cdot p_2^k \leq minPts$ we have $\tilde{O}(p_1^{-k})$ is $\tilde{O}\left(\left(\frac{CP}{minPts}\right)^{\rho}\right)$. □

5 IMPLEMENTATION CHOICES

In this section we highlight implementation-specific choices that we did to speed up the empirical performance of our algorithm.

Details of the hash function. We carry out the empirical analysis under Euclidean distance and use E2LSH [19] as our LSH function. A single hash function $h : \mathbb{R}^d \to \mathbb{Z}$ takes three parameters $a \in \mathbb{R}^d, r \in \mathbb{R}, b \in \mathbb{R}$. For one hash function, $a \sim N(0,1)^d$ is a d-dimensional vector of standard normal random variables, and b is uniformly chosen in the interval $[0, r)$. We use $r = 4\varepsilon$ as recommended [19]. The hash value of a point $p \in \mathbb{R}^d$ is $h_{a,b,r}(p) = \lfloor \frac{a \cdot p + b}{r} \rfloor$, where $a \cdot p$ is the inner product of the vectors a and p. Applying this hash function k times yields a tuple of integers $(H_1, \ldots, H_k) \in \mathbb{Z}^k$. We map these hash values to an integer using universal hashing [16]: Let $P = 2^{61} - 1$ and let random integers $u_1, \ldots, u_k$ be chosen uniformly at random from $\{1, \ldots, P-1\}$. Given $(H_1, \ldots, H_k)$, use $\sum_{1 \leq i \leq k} H_i u_i \mod P$ as key in the hash table.

Details of the merging step. Algorithm 4 details the merging step. For each core point, we conceptually retrieve all the points in all the buckets on the point's best level and check if these two points should belong to the same cluster. Analogously, the work carried out in a bucket is quadratic in its size. Since we are using a union-find data structure to keep track of the *current clustering*, we carry out the work in a single bucket as follows: Let `uf` be the state of the union-find data structure before inspecting the points in bucket B. First, using the `find` operation on `uf`, split up

Table 1: Overview of the datasets.

Dataset	#points (n)	dim. (d)	*minPts*	Baseline
Mnist	60 000	784	100	FAISS
Gist	1 000 000	960	20	—
Glove	1 183 514	100	100	FAISS
Aloi	49 534	27	100	FAISS
Census	299 285	500	650	FAISS
Celeba	202 599	39	200	FAISS
Pamap2	2 872 533	4	100	TPE
Household	2 049 280	7	100	TPE

the points in B into their current clusters. Put all points into an initially empty priority queue PQ and associate with each point the size of its current cluster. While PQ is not empty and there is more than one cluster left, remove the point p with smallest priority and compare it to all points in clusters different than its own. As soon as a point q is found that is at distance at most ε, call `union`(p, q) to merge the two clusterings and stop comparing to other points in q's cluster. Organizing data in this way helps as soon as clusterings evolved while processing different repetitions. Instead of quadratic work in each bucket, the work is at most the product of the individual cluster sizes in a bucket, which can potentially be much smaller.

6 EXPERIMENTAL EVALUATION

This section reports on the results of our experiments. Our code is written in C++ and compiled using the flags `-std=c++17 -O3 -mavx2`. We base our implementation on the code provided by [36] and use the same parallelism framework. Our implementation is available at https://github.com/CamillaOkkels/srrdbscan.

Experimental setup. Experiments were run on a machine with 2x14 core Intel Xeon E5-2690v4 (2.60 GHz) with 512GB RAM using Ubuntu 20.04.6 LTS. All parts of the code are parallelized by using a concurrent hash table (Intel TBB), and splitting up points into batches. We restrict the amount of RAM that can be used to 100 GB and put a time limit of 10 hours on a run for a single parameter.

Hyperparameter choices. Our implementation requires the user to set a bound on the memory consumption (L) and the failure probability (δ). We restrict the memory usage to 5 GB for all datasets except Gist for which we chose to restrict the memory usage to 15 GB. To compare with other baselines, we use $c = 1$ (no approximation) during core point classification, which only puts us at a disadvantage. We choose $\delta \in \{0.01, 0.1, 0.5, 0.9\}$ to control the probability that a core point is missed, cf. Lemma 4.2.

Datasets. Table 1 summarizes the datasets used for the evaluation. We use two popular low-dimensional datasets (Pamap2 and Household) for clustering [44] to verify the performance of our algorithm on low-dimensional datasets. Aloi [26, 38], Census, and Celeba are standard benchmarks in outlier detection [27]; Aloi has been used in clustering analysis [29, 40] and Mnist is a standard dataset used for clustering high-dimensional data [36, 49]. The two high-dimensional datasets Gist and Glove are standard datasets for high-dimensional nearest neighbor search [8] and are the most challenging in terms of their size.

Reproducibility. We developed benchmarking infrastructure to (i) carry out dataset preparation for common datasets used in cluster analysis, (ii) install and run all implementations in Docker containers, and (iii) evaluate the results and produce the plots. This benchmarking and evaluation environment is available at https://github.com/CamillaOkkels/dbscan-benchmark.

Competitors. We compare the proposed implementation (referred to as SRR in the following) to the following baselines: (i) sklearn's DBSCAN[3], (ii) TPEDBSCAN [46], and (iii) SNGDBSCAN [34]. We do not include the projection-based method from [43] because [49] ruled out its scalability even on small high-dimensional datasets. The former is one of the most widely used implementations which speeds up range queries using a tree index data structure and has been shown [41] to be one of the fastest implementations of DBSCAN. The latter two have been described in Section 2 and present the state-of-the-art for grid- and sampling-based approaches. There are no parameters to set for (i) and (ii), and both are exact. For (iii), the user has to manually set the sampling probability and we used the values $\{0.01, 0.05, 0.1, 0.2\}$. We will refer to the competitors as follows: (i) SKL, (ii) TPE, and (iii) SNG. To cope with high-dimensional datasets, we developed our own exact baseline based on the popular FAISS library [21], which provides efficient exact nearest neighbor search with high level of parallelism.[4]

We include IP.LSH [36] in the evaluation, but for the lack of theoretical guarantees omit its results in the head-to-head comparison. A detailed overview of the clustering quality and its running time is provided in Section 6.2.

We do not compare directly to the recent paper by Xu and Pham [49] since they work on inner product spaces and solve other metric spaces by first embedding—distorting the distance—while we are interested in the comparison to DBSCAN in Euclidean space. We remark that they achieve faster clustering times but lack worst-case guarantees, which is a focus of this paper.

Methodology. We follow the methodology of Schubert et al. [44]. For each dataset, a single *minPts* value is fixed. Each implementation is run on a collection of ε values that cover the range from "all points are noise" to "all points belong to a single cluster." As noted in [44] different implementations have different trade-offs regarding the density of the datasets and only testing a wide range of ε values gives an overview of the overall performance of an implementation.

Quality and performance metrics. As quality metric, we measure the *adjusted Rand index* (ARI) [31] of the clustering returned by our algorithm given an exact DBSCAN ground-truth clustering. Values close to 0 (1) indicate a poor (strong) agreement between both solutions. As performance metric, we record the total time needed to cluster the data given some (ε, *minPts*) combination.

Objectives of the experiment. Our experiments are tailored to answer the following questions.

- (Q1) How is the overall running time of the proposed algorithm influenced by the clustering structure of the dataset?
- (Q2) How does the overall running time perform in comparison to other existing baselines?
- (Q3) What is the clustering quality w.r.t. exact DBSCAN results in practice?
- (Q4) How does the proposed multi-level data structure adapt to different choices of ε?

[3] https://scikit-learn.org/stable/modules/clustering.html

[4] We used their exact index with a bruteforce search as the basic search data structure. In the first step, we search for the *minPts*-nearest neighbors to decide whether a point is a core point. Next, we carry out the brute-force search only on core points to find a value K such that the K-th nearest neighbor of each core point is at most at distance ε, starting from the guess $K = minPts$ and doubling in each step. Based on these neighborhoods, we merge labels and assign border points in a final step.

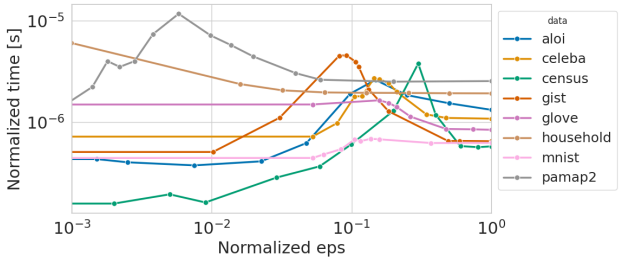
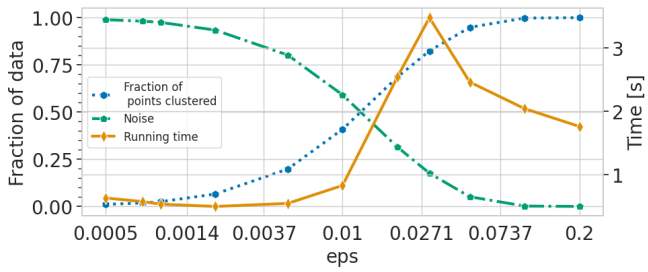
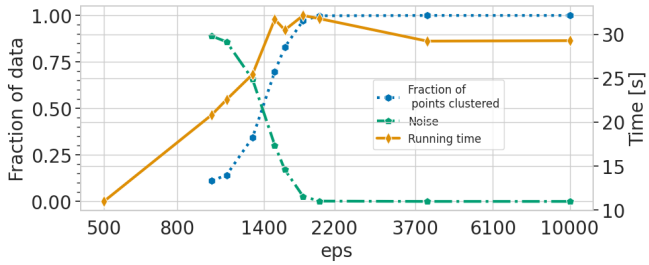


Figure 5: Normalized running times with $\delta = 0.1$; x-axis: normalized ε value using $(\varepsilon - \varepsilon_{\text{min}})/(\varepsilon_{\text{max}} - \varepsilon_{\text{min}})$; y-axis: Running time in seconds normalized by $n \cdot d$.



(a) ALOI



(b) MNIST

Figure 6: Clustering structure and running time for $\delta = 0.1$.

(Q5) How does the approximation factor c influence the running time and result quality?

6.1 Experimental Results

Running time results (Q1 & Q2). Figure 5 summarizes the running time results for running our implementation for $\delta = 0.1$ across all different choices of datasets and ε-values. Because of the dataset diversity regarding their dimensionality d and their size n, the running times are scaled by $d \cdot n$. This normalizes running times to reflect the amount of distance comparisons that have been carried out by the implementation to find the clustering. To account for different distance distributions in the dataset, ε values are scaled between the minimum and maximum value. From the plot we see that higher dimensionality of the dataset does not increase the difficulty of finding the clustering, with no more than a factor 10 difference in normalized running time despite the diversity of tested datasets. Across all datasets, the low-dimensional datasets PAMAP2 and HOUSEHOLD take the longest normalized time to obtain a clustering. Each dataset comes with its own characteristic peak, which poses the question for the relation between the clustering structure and the measured running time (Q1).

Table 2: Results of running time experiments. If method is not exact, fastest configuration with ARI at least .7 is displayed. "$-_{HD}$": dimensionality too high to run implementation. This error is due to a safeguard in the implementation of TPE that restricts the dimension of the input;$-_T/-_M$: 10 hours or 100 GB RAM exceeded respectively. (*) means no baseline ($\delta = 0.1$ for SRR is reported). $|\mathscr{C}|$: number of clusters.

	ε	$\lvert\mathscr{C}\rvert$	Time [s] SRR	SNG	TPE	FAISS	SKL
ALOI	0.0008	2	**0.37**	1.95	$-_{HD}$	21.75	7.69
	0.01	243	0.47	**0.46**	$-_{HD}$	300.02	12.84
	0.1	2	1.44	**1.4**	$-_{HD}$	173.44	49.88
CELEBA	1.0	2291	**4.35**	15.91	$-_{HD}$	323.04	4369.7
	1.6	51	**7.71**	18.37	$-_{HD}$	1078.9	4727.9
	4.0	1	**5.85**	43.83	$-_{HD}$	1616.0	$-_M$
CENSUS	0.03	37	**21.12**	23.4	$-_{HD}$	3768.1	$-_T$
	1.0	1816	**30.96**	93.78	$-_{HD}$	6357.3	$-_T$
	8.0	2	**62.36**	772.88	$-_{HD}$	11823	$-_M$
GIST	0.3*	794	**486.6**	1205.0	$-_{HD}$	$-_T$	$-_T$
	1.3*	27	**3367**	8224.89	$-_{HD}$	$-_T$	$-_T$
	6.0*	1	**625.7**	15627	$-_{HD}$	$-_T$	$-_M$
GLOVE	2.0	8	**176.14**	1064.84	$-_{HD}$	7433.47	$-_T$
	5.0*	11	**165.9**	*2036.14*	$-_{HD}$	$-_T$	$-_T$
	20.0	1	**99.22**	4282.37	$-_{HD}$	18487	$-_M$
MNIST	1000	2	14.41	**13.64**	$-_{HD}$	197.51	3201.3
	1600	7	30.49	**20.05**	$-_{HD}$	683.1	3225.7
	4000	1	**25.2**	42.69	$-_{HD}$	492.7	2946.9
HOUSEHOLD	0.5	69	56.19	$-_M$	**1.13**	$-_T$	$-_M$
	1.0	63	49.43	$-_M$	**1.43**	$-_T$	$-_M$
	500	1	24.24	$-_M$	**0.64**	15609	$-_M$
PAMAP2	0.5	276	15.37	$-_M$	**1.6**	13095	183.86
	3.0	275	80.16	1657.3	**1.49**	$-_T$	319.85
	30.0	3	29.96	3572.3	**1.11**	$-_T$	$-_M$

Table 3: ARI for different epsilon and failure probabilities.

δ	$\varepsilon = 0.0008$ T [s]	ARI	$\varepsilon = 0.01$ T [s]	ARI	$\varepsilon = 0.1$ T [s]	ARI
0.01	0.89	0.9423	1.8	0.9569	2.88	1.0
0.1	0.58	0.9444	0.83	0.9553	2.04	1.0
0.5	0.48	0.9374	0.47	0.9185	1.44	0.9124
0.9	0.37	0.9243	0.44	0.6455	1.04	0.0084

(a) ALOI

δ	$\varepsilon = 1000$ T [s]	ARI	$\varepsilon = 1600$ T [s]	ARI	$\varepsilon = 4000$ T [s]	ARI
0.01	36.11	0.9938	43.4	0.9749	25.2	1.0
0.1	20.82	0.9944	30.49	0.8866	29.22	1.0
0.5	14.41	0.9819	17.62	0.5472	14.77	0.0
0.9	10.34	0.6297	13.9	0.0372	13.49	0.0

(b) MNIST

Figure 6 relates the empirical running time to the clustering structure for two datasets ALOI (medium-dimensional) and MNIST (high-dimensional). We observe distinct phases in the running time for increasing ε-values: When all points are noise (small relative ε), the algorithm has to check each bucket until it confirms that the point is not a core point. In this case, buckets on deeper levels are small since the area is not dense. No work has

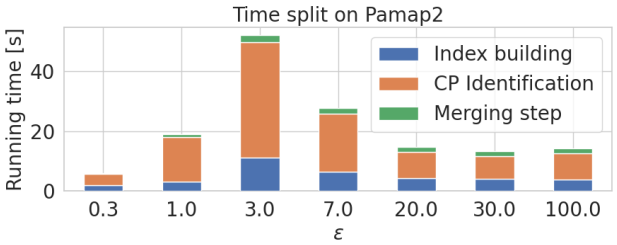


Figure 7: Influence of ε on algorithm steps for failure probability $\delta = 0.1$.

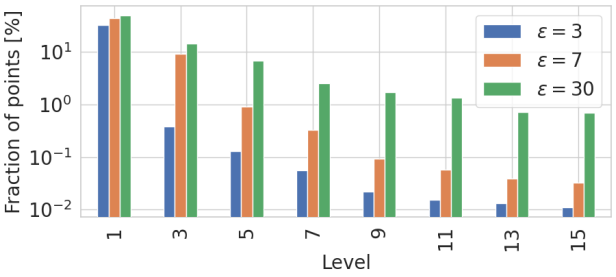


Figure 8: Work per level on Pamap2 for $\varepsilon \in \{3, 7, 30\}$ and $\delta = 0.1$. The y-axis represents the fraction of points that collide with a point, averaged over all dataset points. Note that the y-axis is log-scaled.

to be carried out in the merging step. In the other extreme when all points are core points (large relative ε), it is likely that all points hash into the same bucket, i.e., points will immediately be confirmed as core points; the merging step can merge all points into one cluster based on a single data point as center. The task is most difficult to solve in the setting where clusters have emerged, but they do not all belong to the same clustering yet. The trends observed on these datasets translate to the trends observed in all datasets.

Table 2 compares the running time achieved by our solution to the other baselines, answering (Q2). We make the following observations. As noticed by other authors as well [49], the popular sklearn DBSCAN implementation fails to find a clustering within reasonable memory and time limits. For all datasets except Aloi and Mnist, it fails for some or all ε values, both in terms of running time (e.g., Census) or memory (e.g., Pamap2). For the low-dimensional datasets Household and Pamap2, TPEDBSCAN provides the most efficient solution. It finds the clustering in a few seconds for all parameter choices; our implementation takes a second place and finds all clusterings within a minute. All other competitors are slower and fail for some ε values because of exceeding the time restriction (FAISS) or using too much memory (SNGDBSCAN and SKLEARN). TPEDBSCAN is only useful for low-dimensional datasets.

For medium-sized datasets such as Aloi, Celeba, and MNIST, our implementation and SNGDBSCAN perform best. For high-dimensional and million-sized datasets, only our implementation and SNGDBSCAN finds a clustering within the time and memory restrictions, and our implementation is between 2.4x and 40x faster at finding the clusters. Over all datasets and ε values, for all but GIST the clustering is found in less than 4 minutes. All other baselines cannot reliably find a clustering within 150x this time limit (exceeding 10 hours) or 10x the allowed space (exceeding 100GB RAM).

Influence of δ on clustering quality (Q3). Table 3 showcases the influence of changing δ in our implementation for the datasets Aloi and Mnist, again representing the spectrum of different dataset dimensionality. Recall from Lemma 4.2 that smaller δ increases the repetition count on each level. Fixing an ε value, increasing δ—as expected—improves the running time at cost of lower cluster quality. A δ-value of .5 (50% chance of misclassifying a core point on worst-case input) does not affect the ARI score by much on Aloi, but greatly decreases the clustering quality for Mnist. Since there is no big loss in performance in choosing a small δ-value (running times increased by not more than a factor of 4), these values provide much better result quality at little cost.

Influence of density parameter ε on multi-level data structure (Q4). Figure 7 visualizes the contribution of the different algorithmic subroutines (index building, core point identification, cluster merging) to the total running time on Pamap2 for increasing ε values.[5] When clusters emerge (ε increasing from 0.3 to 7), the time to classify core points is first increasing until $\varepsilon = 3$. The running time of finding the clustering is maximal for this value, and this can be contributed largely to the core point identification step. For larger ε, the contribution of this step decreases since it is easier to verify that a point has at least *minPts* neighbors. In contrast to core point identification, the merging step does not take a significant share of the total running time.

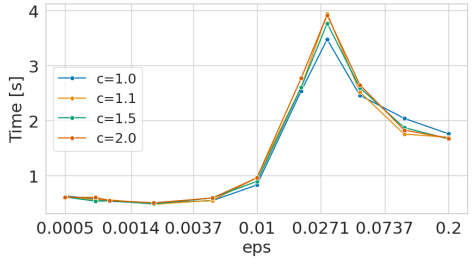
We now focus on the level that is chosen by the proposed data structure. Figure 8 shows the work per level for three different ε values from the discussion above, i.e., the value ω_k in Algorithm 2. Again, the distinct density regions of the clustering have a large influence on the multi-level data structure. For $\varepsilon = 3$, clusters have emerged but points can be well-separated on deeper levels in the data structures such as $K = 15$, where only 0.01% of data points are going to be inspected on average to classify a point. For larger values of ε, the hash function puts more points into the same bucket which allows for quick validation of core points. A lower value of K is chosen in such a case. For example, for $\varepsilon = 7$ and $K = 9$, each point collides with roughly 1% of the data points. Setting these numbers in relation to Figure 7, the core point classification takes less time in that case as for $K = 15$ for the $\varepsilon = 3$.

Influence of approximation factor (Q5). For the following experiment we vary the approximation factor c of SRR (cf. Line 6 in Algorithm 3). For space reasons, we focus on the ALOI and MNIST dataset. As parameters, we choose $c \in \{1.0, 1.1, 1.5, 2.0\}$ and $\delta = 0.1$. From the definition of an approximate DBSCAN* clustering in Section 3, we expect more points to be identified as core points. The impact on running time is less obvious. In the theoretical derivation of the upper bound, cf. Theorem 4.3, a larger c value decreases the running time. Of course, this upper bound is on worst-case input, so a higher approximation may very well lead to a higher running time on real-world datasets. In particular, the processing of the greater number of core points may be more time consuming. An unlucky distribution of core points in the buckets and a large number of emerging clusters could make the merging step more inefficient with more points.

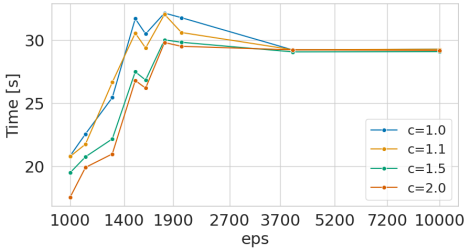
Figure 9 shows the running time as a function of ε for different approximation factors c. For ALOI, there is little to be gained by changing the approximation factor. For MNIST, the running time improved with the approximation factor for almost all values of ε.

Figure 10 illustrates the ARI score of SRR as a function of ε for different approximation factors c. As expected, the overall

[5] The same trends hold for other datasets, but they are most pronounced for Pamap2.

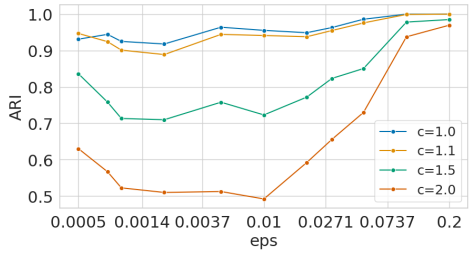


(a) ALOI

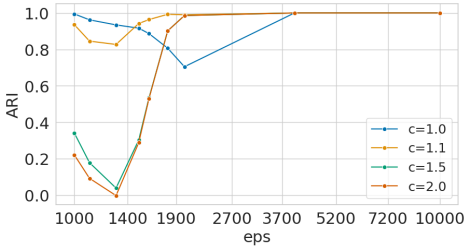


(b) MNIST

Figure 9: Running time as a function of ε for the ALOI and PAMAP2 datasets for different values of c.



(a) ALOI



(b) MNIST

Figure 10: ARI score as a function of ε for the ALOI and PAMAP2 datasets for different approximation factors c.

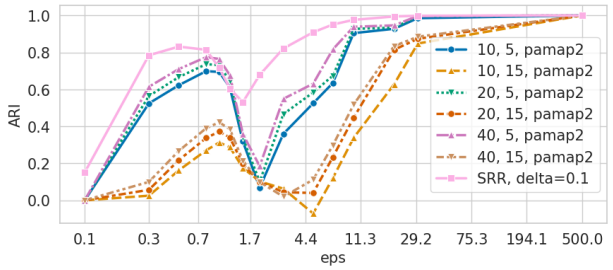
quality decreases as the approximation increases. An exception is for MNIST, for which a value of $c = 1.1$ is actually better for $\varepsilon \geq 1500$.

To summarize, we do not observe a big improvement in running time that would compensate for the loss in quality. We thus recommend to use SRR with $c = 1$.

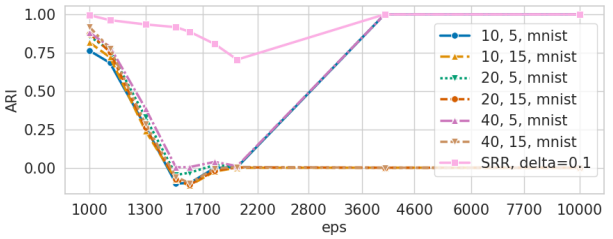
6.2 Comparison to IP.LSH.DBSCAN

In this section we compare our work empirically to the recently suggested IP.LSH method by Keramatian et al. [36].

Overview. Given parameters M and L provided by the user (in addition to the DBSCAN parameters), IP.LSH builds an index for the dataset S that consists of L repetitions. In each repetition, M hash functions are concatenated and each point in S is stored in the bucket identified by the M-tuple of individual hash values.



(a) PAMAP2



(b) MNIST

Figure 11: ARI score for different parameter choices (L, M) for IP.LSH.DBSCAN vs SRRDBSCAN.

This is comparable to building a single level in our data structure with a fixed repetition count that is *not* adjusted automatically to the DBSCAN parameters in combination with the used LSH family.

To identify *core points*, each bucket that contains at least *minPts* many points is checked. The medoid x^* of the points is calculated (the data point closes to the average point of all points that collide in the bucket). This point is marked as a core point if at least *minPts* points are at distance ε in the bucket. All these points at distance at most ε are marked as belonging to the same cluster as x^*. In a second phase, all buckets whose medoid is a core point are inspected again, and distance calculations to other points in the bucket marked as core points, i.e., core points identified in *other buckets*, are carried out. If two points are at distance at most ε, they share a cluster label. As in our implementation, a union-find data structure is employed to keep track of the cluster labeling.

Differences. From a running time perspective, the IP.LSH heuristic alleviates the quadratic time barrier in each bucket; if our implementation and the implementation of IP.LSH would end up using the same level and repetition count, IP.LSH would carry out much less work. However, this improvement comes with great uncertainty about the produced clustering: As stated in [36, Lemma 2], the probability of correctly identifying a core point is extremely low for IP.LSH. This is because it only identifies a single potential point as core point in each bucket. Moreover, for chains of density-reachable core points to get merged into the same cluster, these core points have to be chosen in *other buckets* to be taken into consideration.

Experimental comparison. To confirm whether the uncertainty about the clustering quality can also be observed in practice, we carry out an experimental evaluation using the code provided with [36]. We note that our quality results cannot be compared to [36]: the evaluation provided there focused on the quality of the clustering in terms of predicting known class labels, while we compare to the exact $(\varepsilon, minPts)$-DBSCAN clustering for a diverse range of density parameters ε.

For the experiment, we vary $L \in \{10, 20, 40\}$ and $M \in \{5, 10, 15\}$. We report on the results for Mnist and Pamap2. As we can see from Figure 11, the quality of the resulting clustering varies widely across datasets and ε values. For M equal to 5 and the Mnist dataset, the ARI score is high only for very small density values (almost all points are noise) or very large values (all points belong to the same cluster). In between these values, the quality drops to almost 0. Varying the repetition count L does not alleviate this problem. For $M = 15$ (Figure 11(b)), not enough points collide so core points are never identified and the resulting quality is low.

The trend on the Pamap2 dataset is slightly different. It starts at 0 for low values of ε (note that at $\varepsilon = 0.1$, the Pamap2 dataset actually does have clusters) after which it increases to a local maximum, drops again and then finally increases to 1. Pamap2 has many points, so the local maximum can be explained by the fact that clusters are dense, so there are more chances to find the necessary points to form them. When increasing ε, at some point the clusters that emerged are merged together. This too can be dependent on a single core point being identified. This might be an explanation for the drop we observe around $\varepsilon = 1.5$ to $\varepsilon = 5$.

In terms of the measured running time, the *slowest* clustering time achieved by IP.LSH on MNIST was 1.5s. In comparison, the *fastest* time for SRRDBSCAN of all values of ε with $\delta = 0.1$ was 20.8s. Similarly, while SRRDBSCAN was faster than certain parameter settings of IP.LSH on PAMAP2 for some values of epsilon, the fastest average running time of IP.LSH over all values of epsilon of all parameter settings on PAMAP2 was 2.59s compared an average running time of 48.3s over all ε for SRRDBSCAN with $\delta = 0.1$. IP.LSH finds a clustering much faster, but this has to be seen under the light that the quality with respect to the baseline DBSCAN is unreliable. The comparison is made between the $\delta = 0.1$ run of SRRDBSCAN and the fastest run across the different parameter settings of IP.LSH. As we can see from Tables 3a and 3b, SRRDBSCAN achieves decent quality for all values of epsilon in this setting, whereas the quality of the IP.LSH results is much more inconsistent.

7 CONCLUSION

In this paper we presented SRRDBSCAN, an LSH-based DBSCAN implementation with strong theoretical guarantees on running time and result quality. Through an extensive empirical evaluation, we showed that our implementation provides robust clusterings where other baselines fail because of problems that stem from the curse of dimensionality (exceeding generous running time bounds), too high memory footprint, or too bad clustering quality.

As future work, it would be interesting to apply our methods for finding hierarchical clusterings such as HDBSCAN [14]. Another interesting research agenda would explore the combination of a sampling-based approach such as [34] with LSH.

REFERENCES

[1] Amir Abboud, Vincent Cohen-Addad, and Hussein Houdrouge. 2019. Subquadratic High-Dimensional Hierarchical Clustering. In *NeurIPS*. 11576–11586.

[2] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2007. Robust, Complete, and Efficient Correlation Clustering. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM, 413–418. https://doi.org/10.1137/1.9781611972771.37

[3] Thomas Dybdahl Ahle. 2020. On the Problem of p_1^{-1} in Locality-Sensitive Hashing. In *SISAP (Lecture Notes in Computer Science, Vol. 12440)*. Springer, 85–93.

[4] Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh. 2017. Parameter-free Locality Sensitive Hashing for Spherical Range Reporting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611974782.16

[5] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM* 51, 1 (2008), 117–122.

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. In *NIPS*. 1225–1233.

[7] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pennsylvania, USA) *(SIGMOD '99)*. Association for Computing Machinery, New York, NY, USA, 49–60. https://doi.org/10.1145/304182.304187

[8] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.* 87 (2020).

[9] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When Is "Nearest Neighbor" Meaningful?. In *ICDT (Lecture Notes in Computer Science, Vol. 1540)*. Springer, 217–235.

[10] Christian Böhm, Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. 2004. Density Connected Clustering with Local Subspace Preferences. In *Proceedings of the 4th IEEE International Conference on Data Mining (ICDM 2004), 1-4 November 2004, Brighton, UK*. IEEE Computer Society, 27–34. https://doi.org/10.1109/ICDM.2004.10087

[11] Christian Böhm, Karin Kailing, Peer Kröger, and Arthur Zimek. 2004. Computing Clusters of Correlation Connected Objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, Gerhard Weikum, Arnd Christian König, and Stefan Deßloch (Eds.). ACM, 455–466. https://doi.org/10.1145/1007568.1007620

[12] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*. IEEE, 21–29.

[13] Ricardo J. G. B. Campello, Peer Kröger, Jörg Sander, and Arthur Zimek. 2020. Density-based clustering. *WIREs Data Mining Knowl. Discov.* 10, 2 (2020).

[14] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. 2013. Density-Based Clustering Based on Hierarchical Density Estimates. In *PAKDD (2) (Lecture Notes in Computer Science, Vol. 7819)*. Springer, 160–172.

[15] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. 2015. Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection. *ACM Trans. Knowl. Discov. Data* 10, 1 (2015), 5:1–5:51.

[16] Larry Carter and Mark N. Wegman. 1979. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.* 18, 2 (1979), 143–154.

[17] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. ACM, 380–388.

[18] Flavio Chierichetti and Ravi Kumar. 2015. LSH-Preserving Functions and Their Applications. *J. ACM* 62, 5 (2015), 33:1–33:25.

[19] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*. ACM, 253–262.

[20] Mark de Berg, Ade Gunawan, and Marcel Roeloffzen. 2019. Faster DBSCAN and HDBSCAN in Low-Dimensional Euclidean Spaces. *Int. J. Comput. Geom. Appl.* 29, 1 (2019), 21–47.

[21] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. AAAI Press, 226–231.

[23] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (Melbourne, Victoria, Australia) *(SIGMOD '15)*. Association for Computing Machinery, New York, NY, USA, 519–530. https://doi.org/10.1145/2723372.2737792

[24] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Trans. Database Syst.* 42, 3 (2017), 14:1–14:45.

[25] Junhao Gan and Yufei Tao. 2018. Fast Euclidean OPTICS with Bounded Precision in Low Dimensional Space. In *SIGMOD Conference*. ACM, 1067–1082.

[26] Jan-Mark Geusebroek, Gertjan J. Burghouts, and Arnold W. M. Smeulders. 2005. The Amsterdam Library of Object Images. *Int. J. Comput. Vis.* 61, 1 (2005), 103–112.

[27] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. 2022. ADBench: Anomaly Detection Benchmark. In *NeurIPS*.

[28] Michael E. Houle, Marie Kiermeier, and Arthur Zimek. 2023. Clustering High-Dimensional Data. In *Machine Learning for Data Science Handbook*, L. Rokach, O. Maimon, and E. Shmueli (Eds.). Springer. https://doi.org/10.1007/978-3-031-24628-9_11

[29] Michael E. Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2010. Can Shared-Neighbor Distances Defeat the Curse of Dimensionality?. In *SSDBM (Lecture Notes in Computer Science, Vol. 6187)*. Springer, 482–500.

[30] Yihao Huang, Shangdi Yu, and Julian Shun. 2023. Faster Parallel Exact Density Peaks Clustering. In *ACDA*. SIAM, 49–62.

[31] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification* 2, 1 (1985), 193–218. https://EconPapers.repec.org/RePEc:spr:jclass:v:2:y:1985:i:1:p:193-218

[32] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *STOC*. ACM, 604–613.

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

[34] Heinrich Jiang, Jennifer Jang, and Jakub Lacki. 2020. Faster DBSCAN via subsampled similarity queries. arXiv:2006.06743 [cs.LG]

[35] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. 2004. Density-Connected Subspace Clustering for High-Dimensional Data. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn (Eds.). SIAM, 246–256. https://doi.org/10.1137/1.9781611972740.23

[36] Amir Keramatian, Vincenzo Gulisano, Marina Papatriantafilou, and Philippas Tsigas. 2022. IP.LSH.DBSCAN: Integrated Parallel Density-Based Clustering Through Locality-Sensitive Hashing. In *Euro-Par 2022: Parallel Processing: 28th International Conference on Parallel and Distributed Computing, Glasgow, UK, August 22–26, 2022, Proceedings* (Glasgow, United Kingdom). Springer-Verlag, Berlin, Heidelberg, 268–284. https://doi.org/10.1007/978-3-031-12597-3_17

[37] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe. 2007. Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing. *Knowl. Inf. Syst.* 12, 1 (2007), 25–53. https://doi.org/10.1007/S10115-006-0027-5

[38] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2011. Interpreting and Unifying Outlier Scores. In *SDM*. 13–24.

[39] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2009. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data* 3, 1 (2009), 1:1–1:58. https://doi.org/10.1145/1497577.1497578

[40] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2011. Evaluation of Multiple Clustering Solutions. In *MultiClust@ECML/PKDD (CEUR Workshop Proceedings, Vol. 772)*. CEUR-WS.org, 55–66.

[41] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. 2017. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowl. Inf. Syst.* 52, 2 (2017), 341–378.

[42] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *ICML (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 8748–8763.

[43] Johannes Schneider and Michail Vlachos. 2017. Scalable density-based clustering with quality guarantees using random projections. *Data Min. Knowl. Discov.* 31, 4 (2017), 972–1005.

[44] 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 (2017), 19:1–19:21.

[45] Ye Shiqiu and Zhu Qingsheng. 2019. DBSCAN clustering algorithm based on locality sensitive hashing. In *Journal of Physics: Conference Series*, Vol. 1314. IOP Publishing, 012177.

[46] Yiqiu Wang, Yan Gu, and Julian Shun. 2020. Theoretically-Efficient and Practical Parallel DBSCAN. In *SIGMOD Conference*. ACM, 2555–2571.

[47] Yiqiu Wang, Shangdi Yu, Yan Gu, and Julian Shun. 2021. Fast Parallel Algorithms for Euclidean Minimum Spanning Tree and Hierarchical Spatial Clustering. In *SIGMOD Conference*. ACM, 1982–1995.

[48] Yi-Pu Wu, Jin-Jiang Guo, and Xue-Jie Zhang. 2007. A linear dbscan algorithm based on lsh. In *2007 International Conference on Machine Learning and Cybernetics*, Vol. 5. IEEE, 2608–2614.

[49] Haochuan Xu and Ninh Pham. 2024. Scalable Density-based Clustering with Random Projections. In *NeurIPS*. arXiv:2402.15679 [cs.LG]