# Almost Linear Time Density Level Set Estimation via DBSCAN*

**Hossein Esfandiari**[1], **Vahab Mirrokni**[1], **Peilin Zhong**[2]
[1]Google Research
[2]Columbia University
{esfandiari,mirrokni}@google.com, peilin.zhong@columbia.edu

## Abstract

In this work we focus on designing a fast algorithm for $\lambda$-density level set estimation via DBSCAN clustering. Previous work (Jiang ICML'17, and Jang and Jiang ICML'19) shows that under some natural assumptions DBSCAN and its variant DBSCAN++ can be used to estimate the $\lambda$-density level set with near-optimal Hausdorff distance, i.e., with rate $\widetilde{O}(n^{-1/(2\beta+D)})$. However, to achieve this near-optimal rate, the current fastest DBSCAN algorithm needs near quadratic running time. This running time is not practical for large datasets. Usually when we are working with large datasets we desire linear or almost linear time algorithms. With this motivation, in this work, we present a modified DBSCAN algorithm with near optimal Hausdorff distance for density level set estimation with $\widetilde{O}(n)$ running time. In our empirical study, we show that our algorithm provides significant speedup over the previous algorithms, while achieving comparable solution quality.

## Introduction

Density-based clustering is one of the core problems in data science with a wide range of applications in machine learning, computer vision, and medical imaging among others. Intuitively, in density-based clustering, we have a set of $n$ points in a space and we want to cluster these points by separating the connected dense parts of the space. The celebrated DBSCAN (Ester et al. 1996) is one of the most popular methods for density-based clustering. DBSCAN has been implemented in several data mining tool kits (Team et al. 2013; Hall et al. 2009; Pedregosa et al. 2011; Schubert et al. 2015) and has been successfully used in many applications.

Day after day we are collecting more data and hence have larger datasets to deal with. To the extent that many popular polynomial-time algorithms that we have inherited from the past are not applicable to such massive datasets. For example, an algorithm with a quadratic running time is infeasible when we are dealing with millions of points, let alone hundreds of millions of points. As a result, in the context of large scale algorithm design, it is often desired to have an almost linear time algorithm.

Initially, Ester et al. (Ester et al. 1996) claimed that DBSCAN runs in $O(n \log n)$ time, however, Gunawan and De Berg refuted this claim and showed that this algorithm needs $\Omega(n^2)$ time in the worst-case (Gunawan and de Berg 2013). There are several attempts to improve this running time and get closer the ideal $\tilde{O}(n)$ worst-case running time[1]. However, all of the previous results highly depend on the dimension and hence are only claimed to work well when the dimension of the space is a small constant. For example, in 2D it is possible to implement DBSCAN in $O(n \log n)$ time (de Berg, Gunawan, and Roeloffzen 2017; Gunawan and de Berg 2013). For larger dimensions the state-of-the-art algorithms (Chen, Smid, and Xu 2005; Gan and Tao 2017) require $O(n^{2-O(1/D)})$ time, where $D$ is the dimension of the space. This bound is certainly of great theoretical value, but as $D$ grows, it behaves similar to $O(n^2)$. Moreover, there are some approximation algorithms for DBSCAN that run in $O(n \log n)$ and $O(n)$ when the dimension is a constant (Chen, Smid, and Xu 2005; Gan and Tao 2017). But the running time of these algorithms explicitly depends exponentially on the dimension, which is a drawback.

DBSCAN++ is the state-of-the-art approximation algorithm for DBSCAN that is provably faster than DBSCAN, while interestingly, provides higher quality solutions than DBSCAN in practice (Jang and Jiang 2018). The analysis of DBSCAN++ is based on a standard parameter in level-set analyses called $\beta$-regularity[2]. Specifically, they show that it is possible to approximate DBSCAN in $O(n^{2-\frac{2\beta}{2\beta+D}})$ time. In addition to this, DBSCAN++ can be used to estimate the $\lambda$-density level set with near-optimal Hausdorff distance.

Our main result in this paper is to provide a *provable* $\tilde{O}(n)$-time approximation algorithm for DBSCAN. The algorithm has two main components: The first part is a near linear time core point set construction based on a novel grid density estimation method which unlike previous work avoids $k$-nearest neighbor search which is a bottleneck in previous work. The second part is a graph construction component in which we apply locality sensitive hashing to construct the clusters. Moreover, our algorithm can be used to estimate the $\lambda$-density level set with near-optimal Hausdorff distance in

*All authors contribute equally.



[1]We use $\tilde{O}(f(n))$ to denote $O(f(n) \cdot \log f(n))$.

[2]See Assumtion 7 and the corresponding section for a definition.

$\tilde{O}(n)$ time improving over the $O(n^{2-\frac{2\beta}{2\beta+D}})$ running time of DBSCAN++, which might be of independent interest.

Although, the main goal of this paper is to design a fast algorithm in the classical setting, as a side result, we show how to apply our idea to design a distributed algorithm in the MapReduce model using $\tilde{O}(n)$ work and $O(\log n)$ rounds of computation.

In addition to our theoretical guarantees, we provide an empirical study of the algorithms on real and synthetic data sets. Based on our experiments, our algorithm significantly improves the running time of DBSCAN and DBSCAN++. Moreover, even though our algorithm is an approximation algorithm specifically designed to improve the running time, it achieved comparable quality clusterings to that of DBSCAN and DBSCAN++. Furthermore, we observe that speedup increases significantly as the size of data sets increases, e.g., at $10^8$ points, we reach $100\times$ speedup.

## Algorithm

In this section, we first review the density level set estimation based clustering methods including the classic DBSCAN algorithm (Ester et al. 1996) and its current state-of-the-art approximation DBSCAN++ (Jang and Jiang 2018). Then we introduce our new algorithm — near linear time DBSCAN.

Before delving into more detailed discussions, let us introduce some notation used in this paper. We use $\mathbb{R}_{\geq 0}$ to denote the set of non-negative real numbers. We use $\mathbb{R}^D$ to denote $D$-dimensional Euclidean space. The distance between two points $x, y \in \mathbb{R}^D$ is defined as $d(x,y) := \|x-y\|_2 = \sqrt{\sum_{i=1}^{D}(x_i-y_i)^2}$. We use $\|x-y\|_p$ to denote the $\ell_p$ distance between $x$ and $y$, i.e., $\|x-y\|_p = \left(\sum_{i=1}^{D}|x_i-y_i|^p\right)^{1/p}$. In particular, for $p=\infty$, $\|x-y\|_\infty = \max_i |x_i-y_i|$. We use $[m]$ to denote the set $\{1,2,\cdots,m\}$.

### Clustering via Density Level Set Estimation

The input data is a set of $n$ points $X$ in $D$-dimensional Euclidean space. The goal is to partition $X$ into several clusters. The seminal DBSCAN work (Ester et al. 1996) provides a natural way to generate clusters based on the density of the data points. Suppose $X$ are i.i.d. samples drawn from a distribution $\mathcal{F}$ over $\mathbb{R}^D$. Let $f : \mathcal{X} \to \mathbb{R}_{\geq 0}$ be the density function of $\mathcal{F}$, where $\mathcal{X}$ is the support of $\mathcal{F}$. The definition of $\lambda$-density level set (or $\lambda$-level-set for short) is given in the following.

**Definition 1** (Density level set)**.** *Given $\lambda \geq 0$, the $\lambda$-level-set of $f$ is defined as $L_f(\lambda) := \{x \in \mathcal{X} \mid f(x) \geq \lambda\}$.*

Notice that $\lambda$-level-set may contain multiple connected components in the space. For a given level $\lambda$, an ideal way to partition the data points is via the connected components in the $\lambda$-level-set, i.e., points that fall into the same component are grouped into the same cluster. For data points which are outside of the $\lambda$-level-set, they are put into the cluster corresponding to the closest component in the $\lambda$-level-set. Thus, to generate clusters, the goal becomes to estimate the density level set.

**Algorithm 1** DBSCAN

1: **Inputs:** $X \subset \mathbb{R}^D, \varepsilon, k$
2: Initialize core $C \leftarrow \emptyset$.
3: For each $x \in X$: if $|\{y \in X \mid d(x,y) \leq \varepsilon\}| \geq k$, add $x$ to $C$.
4: Construct a graph $G$: each node corresponds to a point in $X$.
5: For each core point $c \in C$, add an edge in $G$ between $c$ and $x \in X$ which satisfies $d(c,x) \leq \varepsilon$.
6: Return connected components of $G$.

The high level idea of DBSCAN is as the following. Firstly it wants to approximately recognize the points that fall into the density level set. By the definition of the density level set, such points have high density and thus should be in dense regions of the input. Data points in dense regions are called core points. These points are used to estimate the density level set. Next it wants to estimate the connected components of the density level set. Thus, it wants to find connected dense regions. To achieve this goal, it generates a graph by connecting close core points and then it finds connected components of the graph. The remaining data points are clustered by assigning them to the close dense regions. Based on this idea, DBSCAN is described in Algorithm 1.

As shown in Algorithm 1, if there are at least $k$ points in the $\varepsilon$-radius neighborhood of a certain point, it is regarded as a core point. In other words, a core point given by Algorithm 1 has a dense $\varepsilon$-radius neighborhood. To use the output of Algorithm 1 to estimate the connected components of $\lambda$-level-set for a given $\lambda$, one needs to set $k$ and $\varepsilon$ properly according to $\lambda$. For detailed related theoretical analysis, we refer readers to (Jiang 2017; Jang and Jiang 2018). In practice, when $\lambda$ is not explicitly given, $k$ and $\varepsilon$ are usually tuning parameters for obtaining good clustering results.

DBSCAN++ (Jang and Jiang 2018) has a simple modification over DBSCAN. It shows that a downsampled subset of core points still estimates the density level set well. Based on this idea, they first select a subset of $m$ points from the entire dataset and only recognize core points among the sampled $m$ points. Then they follow the remaining steps in the DBSCAN algorithm. In their analysis and experiments, they show that if $m \approx n^{1-2\beta/(2\beta+D)}$, the obtained results have good qualities.

Consider the running time of both algorithms. For DBSCAN, it needs to find $k$-nearest neighbors for each point. Thus, the total running time is $n$ times the time for $k$-nearest neighbor search over $n$ points. For DBSCAN++, it needs to find $k$-nearest neighbors for $m$ sampled points. Thus, the total running time is $m$ times the running time for $k$-nearest neighbor search over $n$ points. As discussed by (Jang and Jiang 2018), although there is a line of work improving the running time of DBSCAN by boosting the $k$-nearest neighbor search in some certain cases (see e.g., (Kumar and Reddy 2016; Vijayalaksmi and Punithavalli 2012; Huang and Bian 2009)), DBSCAN takes near quadratic time and DBSCAN++ takes $O(mn)$ time in general. These running times are far away from near linear runnint time.

## Near Linear Time DBSCAN

We develop a new DBSCAN algorithm which avoids the bottleneck, $k$-nearest neighbor search. By rethinking Algorithm 1, there is a natural question to ask: is there any way beyond looking at the $\varepsilon$-radius neighborhood to distinguish whether a point is in a dense region or not? Suppose data points are drawn from a continuous distribution. If point $x$ has high density, then any point which is close to $x$ must also have high density. Thus, if we look at any small (hyper)cube containing $x$, the probability that a sample falls into the (hyper)cube will be roughly the volume of the (hyper)cube times the density of $x$. Thus, if the data set $X$ is sufficiently large, we will observe a large number of samples in the (hyper)cube, and we can regard such $x$ as a core point. Inspired by this observation, we propose a new core point set construction method in Algorithm 2. Notice that for a point $u \in \mathbb{R}^D$, we use $\lfloor u \rfloor$ to denote a point $v \in \mathbb{Z}^D$ such that each entry of $v$ is obtained by rounding down each entry of $u$, i.e., $\forall i \in \{1, 2, \cdots, D\}, v_i = \lfloor u_i \rfloor$. Roughly speaking, Algorithm 2 partitions the space $\mathbb{R}^D$ into cells where each cell is a (hyper)cube with side length $2\varepsilon$. Notice that two points are in the same cell if and only if they have the same rounded coordinates. Thus, we can easily count the number of points that fall into each cell by sorting and indexing rounded points. If a cell contains at least $k$ points, it will mark every point in this cell as a core point.

Once we obtain core points, the next step is to construct the graph over points. Then another question comes: do we really need to restrict on only connecting the points in the $\varepsilon$-radius neighborhood? If the distance between any two different dense regions are much larger than $\varepsilon$, it is safe to connect core points with distance moderately larger than $\varepsilon$. This observation motivates us to use Locality Sensitive Hashing (LSH), which has broad applications in approximate nearest neighbor search problems. An LSH family is a set of hash functions such that if we draw a hash function from such family, the probability of mapping two close points to the same hash value is high and the probability of mapping two far points to the same hash value is low. We refer readers to a survey (Andoni and Indyk 2008) for more background and literature of LSH. In this work, we use the following LSH in our algorithm.

**Lemma 2.** *Given $\varepsilon \in \mathbb{R}_{\geq 0}$, let $\eta$ be a random variable which has uniform distribution over $[0, 2\varepsilon]$. Let $h : \mathbb{R}^D \to \mathbb{Z}^D$ be a hash function such that $\forall x \in \mathbb{R}^D$, $h(x) := \lfloor \frac{x+\eta \cdot \vec{\mathbf{1}}_D}{2\varepsilon} \rfloor$, where $\vec{\mathbf{1}}_D$ is a $D$-dimensional all-one vector. Then, for any two points $x, y \in \mathbb{R}^D$,*

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

**Algorithm 2** Core Point Set Construction via Rounding

1: **Inputs:** $X \subset \mathbb{R}^D, \varepsilon, k$
2: Construct $\hat{h} : \mathbb{R}^D \to \mathbb{Z}^D$: for $x \in \mathbb{R}^D$, $\hat{h}(x) := \lfloor \frac{x}{2\varepsilon} \rfloor$.
3: Initialize core $C \leftarrow \emptyset$.
4: For $x \in X$, if $|\{y \in X \mid \hat{h}(x) = \hat{h}(y)\}| \geq k$, add $x$ into $C$.
5: Return $C$.

**Algorithm 3** Near Linear time DBSCAN

1: **Inputs:** $X \subset \mathbb{R}^D, t, \varepsilon, k$
2: **//Construct core point set**:
3: Let core $C$ be the output of Algorithm 2.
4: **//Construct a graph over core points**:
5: Draw $t$ independent hash functions $h_1, h_2, \cdots, h_t : \mathbb{R}^D \to \mathbb{Z}^D$, where $h_i$ is constructed as the same as described in Lemma 2: choose $\eta_i \in [0, 2\varepsilon]$ uniformly at random, and $\forall x \in \mathbb{R}^D$, let $h_i(x) := \left\lfloor \frac{x+\eta_i \cdot \vec{\mathbf{1}}_D}{2\varepsilon} \right\rfloor$.
6: Construct a graph $G$: for $i \in [t]$ and for each maximal subset of core points $S \subseteq C$ with the same hash value of $h_i(\cdot)$, choose an arbitrary point in $S$ and connect it to all other points in $S$ in the graph $G$.
7: **//Handle non-core points:**
8: For each non-core point $x \in X \setminus C$, find one arbitrary core point $c \in C$ such that $\exists i \in [t], h_i(c) = h_i(x)$. If such point $c$ exists, connect $x$ to $c$ in $G$.
9: **//Final Clustering:**
10: Return connected components of $G$

*Proof.* Consider two points $x, y \in \mathbb{R}^D$. Fix a coordinate $i \in [D]$. The probability that $\lfloor (x_i+\eta)/2\varepsilon \rfloor \neq \lfloor (y_i+\eta)/2\varepsilon \rfloor$ is at most $|x_i - y_i|/2\varepsilon$. By taking union bound over all coordinates, the probability that $h(x) \neq h(y)$ is at most $\|x - y\|_1/2\varepsilon$, and thus the first claim holds. Now, consider two points $x, y \in \mathbb{R}^D$ with $\|x - y\|_\infty > 2\varepsilon$. There exists a coordinate $i$ such that $x_i - y_i > 2\varepsilon$ which means that for any $\eta \in \mathbb{R}$, $\lfloor (x_i + \eta)/2\varepsilon \rfloor$ can never be equal to $\lfloor (y_i + \eta)/2\varepsilon \rfloor$. Thus, the second claim holds. □

By combining the above LSH with our new core point set construction method (Algortihm 2), we give our near linear time DBSCAN algorithm in Algorithm 3.

**Theorem 3** (Running time of Algorithm 3)**.** *Algorithm 3 can be implemented in $\tilde{O}(ntD)$ time. In particular, if $t = O(\log n)$, it has running time $\widetilde{O}(nD)$ which is near linear in the size of $X$.*

*Proof.* In the stage of core point set construction, we run Algorithm 2. Algorithm 2 can be implemented in $\tilde{O}(nD)$ time: for each $x \in X$, we use $O(D)$ time to compute $\hat{h}(x)$. Then we use sorting which takes $\tilde{O}(nD)$ time to group all points by their rounding value $\hat{h}(x)$. We can determine the core by looking at the size of each group. In the stage of constructing graph over core points, we use $t \cdot O(nD)$ time to compute hash value $h_i(x)$ for each $i \in [t]$ and $x \in X$. Similarly, for each $i \in [t]$, we can use sorting to group all core points by their hash value $h_i(x)$. We create a star in $G$ for each group, i.e., we choose a node in the group and connect it to everyone in the group. Thus, the running time in this stage is at most $t \cdot \tilde{O}(n \cdot D)$. In the stage of handling non-core points, we still use sorting to group points by their hash values. For each core point, we mark the groups it falls in. Since a non-core point only needs to find an arbitrary core point with the same hash value, we only need $O(t)$ time to handle each non-core point. The time in this stage is $\tilde{O}(t \cdot nD)$. The connected components of the graph $G$ can be found in

the time linear in the number of edges which is at most $O(nt)$. Thus, the total running time is at most $\tilde{O}(ntD)$. □

## Theoretical Analysis for Density Level Set Estimation

It was shown that DBSCAN and DBSCAN++ are consistent estimators of the density level sets (Jiang 2017; Jang and Jiang 2018). In this section, we will show that our near linear time DBSCAN algorithm (Algorithm 3) also achieves similar statistical consistency guarantees. As in (Jiang 2017; Jang and Jiang 2018), our density level set estimator is near optimal under Hausdorff distance. Due to space limits, we will put all missing details into Appendix.

The following is a uniform convergence bound (Chaudhuri and Dasgupta 2010). It plays an important role in our analysis.

**Lemma 4** (Theorem 15 of (Chaudhuri and Dasgupta 2010))**.** *Let $X$ be a set of $n$ i.i.d. samples drawn from a distribution $\mathcal{F}$ over $\mathcal{X}$. With probability at least $1-\delta/3$, for any cube $K \subset \mathbb{R}^D$, $\Pr_{x\sim\mathcal{F}}[x \in K] \geq C_{\delta,n}\frac{\sqrt{D\log n}}{n} \Rightarrow |X\cap K| > 0; \Pr_{x\sim\mathcal{F}}[x \in K] \geq \frac{k}{n} + C_{\delta,n}\frac{\sqrt{k}}{n} \Rightarrow |X\cap K| \geq k; \Pr_{x\sim\mathcal{F}}[x \in K] < \frac{k}{n} - C_{\delta,n}\frac{\sqrt{k}}{n} \Rightarrow |X\cap K| < k$, where $C_{\delta,n} = C_0\log(1/\delta)\sqrt{D\log n}$, $C_0$ is a universal constant, and $k \geq C_{\delta,n}$.*

### Regularity Assumptions

We have two assumptions in our theoretical analysis.

**Assumption 5.** *$f$ is continuous and has convex compact support $\mathcal{X} \subseteq \mathbb{R}^D$.*

**Definition 6.** *For $x \in \mathbb{R}^D, A \subseteq \mathbb{R}^D$, define $d(x,A) := \inf_{x'\in A}\|x-x'\|_2$. For $C \subseteq \mathcal{X}, r \geq 0$, define $B(C,r) := \{x \in \mathcal{X} \mid d(x,C) \leq r\}$.*

**Assumption 7** ($\beta$-regularity of level-sets)**.** *Let $\beta \in (0,\infty)$. There exist $C_1, C_2, \lambda_c > 0$ such that $\forall x \in L_f(\lambda-\lambda_c) \setminus L_f(\lambda), C_1\cdot d(x,L_f(\lambda))^\beta \leq \lambda - f(x) \leq C_2\cdot d(x,L_f(\lambda))^\beta$.*

We adopted the same assumptions made by (Jang and Jiang 2018). The first one (Assumption 5) is a natural assumption which asks the distribution to be continuous. The second assumption (Assumption 7) is a standard assumption in level set analysis (see e.g., (Jang and Jiang 2018; Singh et al. 2009)). In high level, it requires that the boundary of the target estimated density level set should have some good properties. One is that the boundary should be salient enough. This is parameterized by the parameter $\beta$. Another is that the distance between two different connected components of the density level set should be far enough. This is described by both $\beta$ and $\lambda_c$ (See the following lemma and corollary).

**Lemma 8.** *Under Assumption 5 and Assumption 7, $\forall x \in B(L_f(\lambda), r_c) \setminus L_f(\lambda)$, $C_1\cdot d(x,L_f(\lambda))^\beta \leq \lambda - f(x) \leq C_2\cdot d(x,L_f(\lambda))^\beta$, where $r_c = \left(\frac{\lambda_c}{C_2}\right)^{1/\beta}$.*

**Corollary 9.** *The distance between any two connected components of $L_f(\lambda)$ is at least $r_c$.*

### Parameters Used in Theoretical Analysis

In this section, we describe our parameter settings. Let $n$ be the size of the point set, i.e., $n = |X|$. Let $\lambda$ be the desired density level. Define $C_{\delta,n} = C_0\sqrt{D\log(n/\delta)}$, where $\delta$ is a confidence parameter, i.e., the desired probability that our guarantees hold is at least $1-\delta$. We suppose that $n$ is sufficiently large. We choose $k$ to be in the range: $\kappa_1\cdot(\log n)^{1.5} \leq k \leq \kappa_2\cdot(\log n)^{\frac{D}{2\beta+D}} n^{\frac{2\beta}{2\beta+D}}$, for some sufficiently large parameter $\kappa_1$ and sufficiently small parameter $\kappa_2$, where $\kappa_1, \kappa_2$ only depends on $D, \lambda, \delta$ and the density function $f(\cdot)$. We choose $\varepsilon = \frac{1}{2}\left(\frac{k}{n\cdot\lambda\cdot(1-2C_{\delta,n}/\sqrt{k})}\right)^{1/D}$ and let $t = C_t\log(n/\delta)$ for some sufficiently large universal constant $C_t$.

### Density Level Set Estimation

In this section, we show that the output of our new core point set construction, Algorithm 2, is indeed a good estimation to the desired density level set. Firstly, we show that if a sampled point is too far away from the desired density level set, it will not be added into the core.

**Lemma 10.** *$\forall x \in X$, if $d(x,L_f(\lambda)) \geq 2\left(\frac{\lambda}{C_1}\cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{\frac{1}{\beta}}$, then $x$ is not in the core.*

*Proof.* If $y \in X$ satisfies $\hat{h}(y) = \hat{h}(x)$, then $\|x-y\|_2 \leq \sqrt{D}\|x-y\|_\infty \leq 2\sqrt{D}\varepsilon$. We have $d(y,L_f(\lambda)) \geq d(x,L_f(\lambda)) - 2\sqrt{D}\varepsilon \geq d(x,L_f(\lambda))/2$, where the last inequality follows from that $\varepsilon$ is properly chosen and $\kappa_2$ is sufficiently small. By Assumption 7, we have $f(y) \leq \lambda - \lambda\cdot 10C_{\delta,n}/\sqrt{k}$. Then, $\int_{\mathcal{X}} f(z)\cdot\mathbf{1}\left(\hat{h}(z)=\hat{h}(x)\right)\mathrm{d}z \leq (2\varepsilon)^D\cdot(\lambda - \lambda\cdot 10C_{\delta,n}/\sqrt{k}) \leq \frac{k}{n\lambda(1-2C_{\delta,n}/\sqrt{k})}\cdot\lambda(1 - 10C_{\delta,n}/\sqrt{k}) \leq \frac{k}{n} - \frac{\sqrt{k}}{n}\cdot 8C_{\delta,n}$. According to Lemma 4, we know that $|\{y\in X \mid \hat{h}(x)=\hat{h}(y)\}| < k$ which means that $x$ will not be added into the core. □

Then we show that if a sampled point is in the density level set, or it is sufficiently close to the density level set, it must be added into the core by Algorithm 2.

**Lemma 11.** *$\forall x \in X$, if $d(x,L_f(\lambda)) \leq \frac{1}{2}\left(\frac{\lambda}{C_2}\cdot\frac{C_{\delta,n}}{\sqrt{k}}\right)^{\frac{1}{\beta}}$, then $x$ will be added into the core.*

*Proof.* If $y \in X$ satisfies $\hat{h}(y) = \hat{h}(x)$, then $\|x-y\|_2 \leq \sqrt{D}\|x-y\|_\infty \leq 2\sqrt{D}\varepsilon$. Thus, $d(y,L_f(\lambda)) \leq d(x,L_f(\lambda)) + 2\sqrt{D}\varepsilon \leq \left(\frac{\lambda}{C_2}\cdot C_{\delta,n}/\sqrt{k}\right)^{1/\beta}$, where the last inequality follows from that $\varepsilon$ is properly chosen and $\kappa_2$ is sufficiently small. By Assumption 7, we have $f(y) \geq \lambda - \lambda C_{\delta,n}/\sqrt{k}$. Then, $\int_{\mathcal{X}} f(z)\cdot\mathbf{1}(\hat{h}(z)=\hat{h}(x))\mathrm{d}z \geq (2\varepsilon)^D\cdot\left(\lambda - \lambda C_{\delta,n}/\sqrt{k}\right) \geq \frac{k}{n\lambda(1-2C_{\delta,n}/\sqrt{k})}\cdot\lambda(1-C_{\delta,n}/\sqrt{k}) \geq \frac{k}{n} + C_{\delta,n}\frac{\sqrt{k}}{n}$. According to Lemma 4, we know that $|\{y\in X \mid \hat{h}(x)=\hat{h}(y)\}| \geq k$ which means that $x$ will be added into the core. □

Next, we show that for any point which is in the desired density level set, there is a close sampled point which will be added into the core.

**Lemma 12.** *$\forall x \in L_f(\lambda)$, $\exists$ a core point $x' \in C$ such that $\|x - x'\|_2 \le \frac{\varepsilon}{2} \le \frac{1}{2}\left(\frac{\lambda}{C_2} \cdot C_{\delta,n}/\sqrt{k}\right)^{1/\beta}$.*

*Proof.* Let $r_0 = \frac{1}{2}\left(\frac{2C_{\delta,n}\sqrt{D\log n}}{n\lambda}\right)^{1/D}$. We have: $\int_{\mathcal{X}} f(z) \cdot \mathbf{1}(\|z - x\|_\infty \le r_0)\mathrm{d}z \ge (2r_0)^D(\lambda - C_2 \cdot r_0^\beta) \ge (2r_0)^D\lambda/2 \ge \frac{C_{\delta,n}\sqrt{D\log n}}{n}$. By Lemma 4, there is a point $x' \in X$ such that $\|x - x'\|_\infty \le r_0$. Since $\kappa_1$ is sufficiently large, by analyzing the range of $k$, we have $r_0 \le \frac{\varepsilon}{2\sqrt{D}}$. Since $\varepsilon$ is chosen properly and $\kappa_2$ is sufficiently small, we have $\frac{\varepsilon}{2} \le \frac{1}{2} \cdot \left(\frac{\lambda}{C_2} \cdot C_{\delta,n}/\sqrt{k}\right)^{1/\beta}$. By Lemma 11, $x'$ is in $C$. □

The Hausdorff Distance is defined as $\mathrm{d}_{\text{Haus}}(A, A') = \max\{\sup_{x\in A}\mathrm{dist}(x, A'), \sup_{x'\in A'}\mathrm{dist}(x', A)\}$. Now, we are able to bound the Hausdorff error of using the core returned by Algorithm 2 to estimate the desired density level set. The theorem follows directly from Lemma 12 and Lemma 10.

**Theorem 13.** *Let $C$ be the output of Algorithm 2, then, $d_{Haus}(C, L_f(\lambda)) \le 2\left(\frac{\lambda}{C_1} \cdot 10\frac{C_{\delta,n}}{\sqrt{k}}\right)^{1/\beta}$.*

**Remark 14.** *If we choose maximum possible $k$, the above quantity is at most $\tilde{O}\left(\kappa_3 \cdot n^{-\frac{1}{2\beta+D}}\right)$, where $\kappa_3$ is a parameter only depends on $D, \delta, \lambda$ and the density function $f$. This matches the lower bound shown in Theorem 4 of (Tsybakov et al. 1997). Thus, our density level set estimation is near optimal.*

**Connected components estimation.** After obtaining the core point set, the next step in Algorithm 3 is to construct the graph over the data points. We show that there is actually a one-to-one correspondence between the connected components of $\lambda$-density level sets and the connected components of the graph constructed by Algorithm 3. We defer these results to Appendix.

## Efficient DBSCAN in Distributed Models

Observe that all operations in our algorithm are highly parallelizable. In theory, beyond classic sequential setting, our algorithm can also be efficiently implemented in parallel setting and distributed setting in general. As one example, our algorithm can be implemented in MapReduce (Dean and Ghemawat 2008). A commonly studied formal model for MapReduce is the (Massively Parallel Computation) MPC model which is introduced by (Karloff, Suri, and Vassilvitskii 2010; Goodrich, Sitchinava, and Zhang 2011; Beame, Koutris, and Suciu 2017). In this model, there are $p$ machines where each has local memory size $s = O(N^\delta)$. Here $N$ is the total size of the input data and $\delta \in (0, 1)$ is a constant. It implies that the local memory of each machine is sublinear in the input size. In the most restricted case, the total space $p \cdot s$ in the system is $\tilde{O}(N)$ where $N$ is the total size of the input data, i.e., the total space is slightly larger than the input size. Before the computation starts, the input is distributed on $\Theta(N/s)$ input machines. The computation proceeds in rounds. In each round, a machine does some local computation and sends messages to other machines at the end of the round. In a round, the total communication of a machine must be bounded by its local memory size $s$. In the next round, each machine only holds the received messages in its local memory. At the end of the computation, the output data is distributed on the output machines. The goal is to design an algorithm with small number of rounds.

**Theorem 15** (DBSCAN in MapReduce)**.** *Except final clustering stage, Algorithm 3 can be implemented in the MPC model with $O(1)$ number of rounds and $\tilde{O}(ntD)$ total space. In particular, if $t = O(\log n)$, the total space needed is $\tilde{O}(nD)$.*

**Complete Algorithm 3 in the MPC model.** The only stage remaining is to compute connected components of the constructed graph $G$. In the MPC model, this can be done in $\sim \log$(Diameter of the graph) number of rounds (Andoni et al. 2018; Behnezhad et al. 2019a). In a stronger Adaptive Massively Parallel Computation (AMPC) model (Behnezhad et al. 2019b), connected components can be computed in $O(\log\log n)$ rounds. By combining with Theorem 15, we can run our DBSCAN clustering algorithm in MPC model with $O(\log n)$ rounds and in AMPC model with $O(\log\log n)$ rounds. In general, if connected components can be computed in MPC/AMPC model in $R$ rounds, our algorithm can be implemented in MPC/AMPC model in $R + O(1)$ rounds.

## Experiments

**Setup.** We evaluated our algorithm, Algorithm 3, on both synthetic and real world datasets. We implemented DBSCAN, DBSCAN++ (Jang and Jiang 2018) and our algorithm. We implemented two versions (indexed as DSv1 and DSv2 respectively) of DBSCAN, where one is exactly the same as Algorithm 1 and another is a natural variant but has a better accuracy in practice (See Appendix for more details). There are also two versions of DBSCAN++, where one (indexed as

| Datasets | $n$ | $D$ | $c$ |
|---|---|---|---|
| (A) iris | 150 | 4 | 3 |
| (B) wine | 178 | 13 | 3 |
| (C) spam | 4601 | 57 | 2 |
| (D) images | 210 | 19 | 7 |
| (E) MNIST | 60000 | 20 | 10 |
| (F) Libras | 360 | 90 | 15 |
| (G) mobile | 2000 | 20 | 4 |
| (H) zoo | 101 | 16 | 7 |
| (I) seeds | 210 | 7 | 3 |
| (J) letters | 20000 | 16 | 26 |
| (K) phonemes | 4509 | 256 | 5 |
| (L) YouTube8M | 764321 | 128 | 3 |

Table 1: Real world datasets summary. In the table, $n$ denotes the number of data points, $D$ denotes the dimension, and $c$ denotes the number of ground truth clusters. We index datasets from (A) to (L).

| | DSv1 | DSv2 | DS++ k-ctr | DS++ unif | Ours |
|---|---|---|---|---|---|
| (A) | 0.5681 | **0.6899** | 0.6634 | 0.5823±0.07 | 0.7066±0.01 |
| | 0.7316 | **0.7316** | 0.7508 | 0.6512±0.07 | **0.7316**±0.00 |
| (B) | 0.2883 | 0.3122 | 0.3694 | **0.3615**±0.02 | 0.3366±0.02 |
| | 0.3636 | 0.4006 | 0.4246 | **0.4177**±0.01 | 0.4037±0.02 |
| (C) | 0.1301 | 0.1453 | 0.1304 | **0.1404**±0.00 | 0.1128±0.00 |
| | 0.0455 | 0.1076 | 0.1210 | **0.1080**±0.00 | 0.0682±0.00 |
| (D) | 0.3244 | 0.3578 | **0.3783** | 0.3477±0.02 | 0.3979±0.01 |
| | 0.4560 | 0.6238 | 0.5380 | 0.5168±0.02 | **0.5828**±0.00 |
| (E) | 0.1950 | 0.2204 | 0.2356 | **0.2561**±0.00 | 0.2757±0.01 |
| | 0.0760 | 0.4203 | **0.4581** | 0.4779±0.00 | 0.4223±0.00 |
| (F) | 0.1420 | 0.1173 | **0.1883** | 0.1602±0.01 | 0.2311±0.01 |
| | 0.2037 | 0.2722 | **0.4461** | 0.3505±0.02 | 0.4554±0.01 |
| (G) | 0.0192 | **0.3559** | 0.4087 | 0.2989±0.02 | 0.2005±0.01 |
| | 0.0618 | **0.4550** | 0.4597 | 0.4143±0.01 | 0.3196±0.00 |
| (H) | 0.7169 | 0.6814 | **0.7171** | 0.5775±0.06 | 0.7568±0.00 |
| | 0.6832 | 0.6921 | **0.7375** | 0.6140±0.02 | 0.8262±0.00 |
| (I) | 0.3893 | 0.6948 | 0.5738 | 0.5803±0.04 | **0.6408**±0.02 |
| | 0.3647 | 0.6787 | 0.6032 | 0.6067±0.03 | **0.6230**±0.01 |
| (J) | 0.1096 | 0.1651 | 0.1185 | 0.1117±0.00 | **0.1390**±0.01 |
| | 0.3457 | 0.5004 | **0.4926** | 0.4875±0.00 | 0.4664±0.00 |
| (K) | 0.4577 | 0.7344 | 0.3558 | **0.6482**±0.02 | 0.3666±0.02 |
| | 0.3297 | 0.8229 | 0.5506 | **0.7343**±0.02 | 0.2858±0.02 |

Table 2: Scores of algorithms on real world datasets (A)-(K). For each dataset, the first row corresponds to Adjusted Rand Index scores and the second row corresponds to Adjusted Mutual Information. In each row, the highest score is shown in bold and the second highest score is underlined. DBSCAN++ with uniform initialization and our algorithm are randomized algorithms. We took 10 runs to report the standard error. Other algorithms are deterministic. As we can see that our algorithm has the highest score on 7 metrics and has top-2 score on 12 metrics. For each compared algorithm, our algorithm has better scores on at least half of the total 22 metrics.

DS++ unif) uses uniform sampling to choose $m$ samples in its initialization stage, and another (indexed as DS++ k-ctr) uses greedy k-center initialization (Jang and Jiang 2018) to choose $m$ samples. For each version of DBSCAN and DBSCAN++, we implemented two variants. The first variant uses KDTree (Bentley 1975) to handle $k$-nearest neighbor search while the second variant uses brute-force[3]. We ran all of them on 11 real world datasets that are also used in (Jang and Jiang 2018). In addition, We ran our algorithm and all implemented versions of DBSCAN++ on an additional large real world dataset for the scalability test. A brief summary of these datasets is shown in Table 1. Dataset (E) MNIST (LeCun et al. 1998) contains 10 classes of handwritten digits. Each original data point in MNIST has $28 \times 28 = 784$ dimensions. We use PCA to reduce the dimension to 20. Dataset (G) mobile is a dataset for mobile price classification. Based on different price ranges, data points are partitioned into 4 clusters. This dataset is available on Kaggle[4]. Dataset (K) phonemes (Friedman, Hastie, and Tibshirani 2001) contains log periodograms of spoken phonemes. Based on different phonemes, data points are partitioned into 5 clusters. Dataset

[3]For KDTree part, we used public codes shared here: https://github.com/crvs/KDTree.

[4]See https://www.kaggle.com/.

| | DS++ k-ctr | | DS++ unif | | Ours |
|---|---|---|---|---|---|
| use kd-tree | N | Y | N | Y | N/A |
| (L) | 9862.41 | 8982.18 | 5146.73 | 7260.41 | **56.69** |

Table 3: Running time for real world dataset (L). We ran our algorithm and all implemented versions of DBSCAN++ on the dataset (L). We report the running time (in seconds) for each listed algorithm on the dataset (L). For each version of DBSCAN++, we report both running time of using and without using KDTree for handling $k$-nearest neighbor search.

(L) YouTube8M[5] contains video features (Abu-El-Haija et al. 2016). The full dataset contains 31GB features. We selected the first 3 classes of videos (Game, Vehicle, Video Game) as 3 clusters. We choose the 128 dimensional audio features as data points. Other datasets are available on UCI repository (Dua and Graff 2017). Readers may find detailed descriptions of these datasets there. All algorithms are implemented in C++. Adjusted Rand Index scores and Adjusted Mutual Information scores are evaluated via the python3 package scikit-learn with version 0.23.1. All experiments are performed on a machine with 32G main memory and Intel(R) Xeon(R) CPU @ 2.30GHz. Our synthetic datasets are Gaussian mixtures in Euclidean space with various dimensions and various number of points. Each dataset has 3 equal-size clusters.

In all experiments, we fixed $k = 10$. For each clustering task, DBSCAN++ needs to subsample $m$ data points and computes core points among these subsamples. As suggested by (Jang and Jiang 2018), we set $m = \lfloor 0.1 \cdot n^{D/(D+4)} \rfloor$ for all experiments. For our algorithm, we set $t$, the number of hash functions, to be $\lfloor 2 \cdot \ln(n) \rfloor$. All experiments are done on a single machine. All programs are in single thread mode. For experiments on synthetic datasets, we fix $\varepsilon = 3 \cdot \sqrt{D}$ for all algorithms. For more detailed experiment setup, we refer readers to Appendix.

**Evaluation on small real world datasets.** We evaluate both accuracy and speed for all implemented algorithms on datasets (A) to (K) described in Table 1. The accuracy is evaluated under metrics Adjusted Rand Index scores and Adjusted Mutual Information (Vinh, Epps, and Bailey 2010). We ran the same optimal tuning procedure to find the best $\varepsilon$ for each algorithm and reported the corresponding scores. As shown in Table 2 and Table 4, the qualities of the clustering results obtained by our algorithm are as good as DBSCAN and DBSCAN++ while the running time of our algorithm is always the fastest.

**Speed-up for large-scale datasets.** Table 4 shows that DBSCAN++ and our algorithm can gain significant speed-up even when $n$ is about thousands. It is natural to ask what the behavior of DBSCAN++ and our algorithm is when the size of the dataset becomes much larger. We first test our algorithm and all implemented versions of DBSCAN++ on

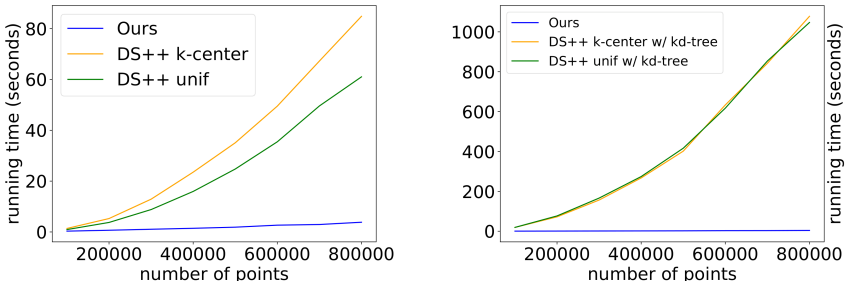
[5]See http://research.google.com/youtube8m/download.html.

| | DSv1 | | DSv2 | | DS++ k-ctr | | DS++ unif | | Ours |
|---|---|---|---|---|---|---|---|---|---|
| use kd-tree | N | Y | N | Y | N | Y | N | Y | N/A |
| (C) | 2.33 ±0.00 | 7.62 ±0.01 | 1.34 ±0.00 | 2.23 ±0.01 | 0.45 ±0.00 | 0.45 ±0.00 | 0.23 ±0.00 | 0.39 ±0.00 | **0.11** ±0.00 |
| (E) | 73.65 ±0.20 | 84.20 ±0.28 | 89.63 ±1.43 | 225.51 ±2.65 | 2.55 ±0.01 | 9.58 ±0.07 | 2.18 ±0.01 | 13.03 ±0.06 | **1.01** ±0.01 |
| (F) | 0.02 ±0.00 | 0.06 ±0.00 | 0.02 ±0.00 | 0.07 ±0.00 | 0.01 ±0.00 | 0.02 ±0.00 | 0.01 ±0.00 | 0.02 ±0.00 | **0.00** ±0.00 |
| (G) | 0.10 ±0.00 | 0.48 ±0.00 | 0.09 ±0.00 | 0.45 ±0.00 | 0.01 ±0.00 | 0.04 ±0.00 | 0.00 ±0.00 | 0.03 ±0.00 | **0.00** ±0.00 |
| (J) | 7.97 ±0.01 | 7.83 ±0.15 | 7.83 ±0.15 | 17.12 ±0.04 | 0.54 ±0.00 | 2.15 ±0.00 | 0.34 ±0.00 | 1.31 ±0.01 | **0.28** ±0.00 |
| (K) | 10.00 ±0.04 | 27.52 ±0.29 | 9.46 ±0.01 | 35.46 ±0.14 | 1.34 ±0.00 | 2.93 ±0.01 | 1.17 ±0.01 | 3.95 ±0.01 | **0.23** ±0.01 |

Table 4: Running time for real world datasets (A)-(K). We ran each algorithm on each dataset 10 times. We report the mean running time (in seconds) and standard errors for each algorithm on each dataset. For datasets (A), (B), (D), (H), (I), every implemented algorithm can finish in at most 0.01 seconds and has standard error 0.00. For each version of DBSCAN and DBSCAN++, we report both running time of using and without using KDTree for handling $k$-nearest neighbor search. In all experiments, our algorithm is the fastest one.

| | DS++ k-ctr | | DS++ unif | | Ours | Data generation |
|---|---|---|---|---|---|---|
| use kd-tree | N | Y | N | Y | N/A | N/A |
| $n = 10^6, D = 10$ | 153.77 | 1738.85 | 118.46 | 1671.93 | **4.35** | 0.47 |
| $n = 10^6, D = 100$ | 40043.24 | >1day | 30643.61 | >1day | **22.89** | 3.93 |
| $n = 10^6, D = 300$ | >1day | >1day | >1day | >1day | **113.82** | 11.66 |
| $n = 10^7, D = 10$ | >1day | >1day | >1day | >1day | **56.08** | 5.54 |
| $n = 10^8, D = 10$ | >1day | >1day | >1day | >1day | **708.121** | 48.58 |

Table 5: Running time (in seconds) for extremely large synthetic datasets. For synthetic datasets with extremely large size, our algorithm only needs about $10\times$ the time needed to generate the data. Our algorithm can have more than $100\times$ speed-up comparing with DBSCAN++. In addition, in all experiments, all algorithms have Adjusted Rand Index score 1.0 and Adjusted Mutual Information score 1.0.




(a) (b)

Figure 1: Running time vs. number of points. In both above figures, $x$-axis corresponds to the number of points in the dataset, and $y$-axis corresponds to the running time. In (a) and (b), we can see that the running time of our algorithm is much faster than DBSCAN++. For these synthetic datasets, KDTree cannot help in improving the running time of DBSCAN++. It even introduces a large overhead. The running time of our algorithm grows almost linearly while the running time of each version of DBSCAN++ grows much faster than linear. In addition, in all experiments, all algorithms have Adjusted Rand Index score 1.0 and Adjusted Mutual Information score 1.0.

the large real world dataset (L) which has $n = 764321$ and $D = 128$. We choose $\varepsilon = 1.5$ and the running time of each algorithm is shown in Table 3. Since all versions of DBSCAN++ took several hours for even one run on the dataset (L), we cannot afford to run optimal tunning procedure to find the best $\varepsilon$ for DBSCAN++ on (L). Thus, we did not compare the accuracy on (L). Then we test our algorithm and all implemented versions of DBSCAN++ on two bathces of synthetic datasets. In the first batch of synthetic experiments, we generated mixture of Gaussians in 10-dimensional space with 3 clusters. We enumerate $n$ from $10^5$ to $8 \cdot 10^5$ and see the change of running time of DBSCAN++ and our algorithm when $n$ increases. In the second batch of synthetic experiments, we ran DBSCAN++ and our algorithm for extremely large number of points and larger dimensions. For all synthetic experiments, all evaluated algorithms always recover 3 clusters perfectly. The running time for $n \in [10^5, 8 \cdot 10^5]$ is shown in Figure 1. The running time for extremely large synthetic dataset is given in Table 5.

## Acknowledgements

Peilin Zhong is supported in part by NSF grants CCF-1740833, CCF-1703925, CCF-1714818 and CCF-1822809 and a Google Ph.D. Fellowship.

## References

Abu-El-Haija, S.; Kothari, N.; Lee, J.; Natsev, P.; Toderici, G.; Varadarajan, B.; and Vijayanarasimhan, S. 2016. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675* .

Andoni, A.; and Indyk, P. 2008. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *COMMUNICATIONS OF THE ACM* 51(1): 117.

Andoni, A.; Song, Z.; Stein, C.; Wang, Z.; and Zhong, P. 2018. Parallel graph connectivity in log diameter rounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, 674–685. IEEE.

Beame, P.; Koutris, P.; and Suciu, D. 2017. Communication steps for parallel query processing. *Journal of the ACM (JACM)* 64(6): 1–58.

Behnezhad, S.; Dhulipala, L.; Esfandiari, H.; Lacki, J.; and Mirrokni, V. 2019a. Near-optimal massively parallel graph connectivity. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 1615–1636. IEEE.

Behnezhad, S.; Dhulipala, L.; Esfandiari, H.; Łącki, J.; Mirrokni, V.; and Schudy, W. 2019b. Massively parallel computation via remote memory access. In *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, 59–68.

Bentley, J. L. 1975. Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9): 509–517.

Chaudhuri, K.; and Dasgupta, S. 2010. Rates of convergence for the cluster tree. In *Advances in neural information processing systems*, 343–351.

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

de Berg, M.; Gunawan, A.; and Roeloffzen, M. 2017. Faster DB-scan and HDB-scan in low-dimensional Euclidean spaces. *arXiv preprint arXiv:1702.08607* .

Dean, J.; and Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM* 51(1): 107–113.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. URL http://archive.ics.uci.edu/ml.

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

Friedman, J.; Hastie, T.; and Tibshirani, R. 2001. *The elements of statistical learning*, volume 1. Springer series in statistics New York.

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

Goodrich, M. T. 1999. Communication-efficient parallel sorting. *SIAM Journal on Computing* 29(2): 416–432.

Goodrich, M. T.; Sitchinava, N.; and Zhang, Q. 2011. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, 374–383. Springer.

Gunawan, A.; and de Berg, M. 2013. A faster algorithm for DBSCAN. *Master's thesis. Eindhoven University of Technology, the Netherlands* .

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

Huang, M.; and Bian, F. 2009. A grid and density based fast spatial clustering algorithm. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 4, 260–263. IEEE.

Jang, J.; and Jiang, H. 2018. DBSCAN++: Towards fast and scalable density clustering. *arXiv preprint arXiv:1810.13105* .

Jiang, H. 2017. Density level set estimation on manifolds with dbscan. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1684–1693. JMLR. org.

Karloff, H.; Suri, S.; and Vassilvitskii, S. 2010. A model of computation for MapReduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, 938–948. SIAM.

Kpotufe, S.; and Von Luxburg, U. 2011. Pruning nearest neighbor cluster trees. *arXiv preprint arXiv:1105.0540* .

Kumar, K. M.; and Reddy, A. R. M. 2016. A fast DBSCAN clustering algorithm by accelerating neighbor searching using Groups method. *Pattern Recognition* 58: 39–48.

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

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

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

Singh, A.; Scott, C.; Nowak, R.; et al. 2009. Adaptive hausdorff estimation of density level sets. *The Annals of Statistics* 37(5B): 2760–2782.

Team, R. C.; et al. 2013. R: A language and environment for statistical computing .

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

Vijayalaksmi, S.; and Punithavalli, M. 2012. A fast approach to clustering datasets using dbscan and pruning algorithms. *International Journal of Computer Applications* 60(14).

Vinh, N. X.; Epps, J.; and Bailey, J. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research* 11: 2837–2854.