Faster DBSCAN via subsampled similarity queries

Heinrich Jiang*
Google Research
heinrichj@google.com

Jennifer Jang*
Waymo
jangj@waymo.com

Jakub Łącki
Google Research
jlacki@google.com

Abstract

DBSCAN is a popular density-based clustering algorithm. It computes the ϵ-neighborhood graph of a dataset and uses the connected components of the high-degree nodes to decide the clusters. However, the full neighborhood graph may be too costly to compute with a worst-case complexity of $O(n^2)$. In this paper, we propose a simple variant called SNG-DBSCAN, which clusters based on a subsampled ϵ-neighborhood graph, only requires access to similarity queries for pairs of points and in particular avoids any complex data structures which need the embeddings of the data points themselves. The runtime of the procedure is $O(sn^2)$, where s is the sampling rate. We show under some natural theoretical assumptions that $s \approx \log n/n$ is sufficient for statistical cluster recovery guarantees leading to an $O(n \log n)$ complexity. We provide an extensive experimental analysis showing that on large datasets, one can subsample as little as 0.1% of the neighborhood graph, leading to as much as over 200x speedup and 250x reduction in RAM consumption compared to scikit-learn's implementation of DBSCAN, while still maintaining competitive clustering performance.

1 Introduction

DBSCAN [13] is a popular density-based clustering algorithm which has had a wide impact on machine learning and data mining. Recent applications include superpixel segmentation [40], object tracking and detection in self-driving [54, 19], wireless networks [12, 56], GPS [10, 36], social network analysis [29, 58], urban planning [14, 38], and medical imaging [48, 3]. The clusters that DBSCAN discovers are based on the connected components of the neighborhood graph of the data points of sufficiently high density (i.e. those with a sufficiently high number of data points in their neighborhood), where the neighborhood radius and the density threshold are the hyperparameters.

One of the main differences of density-based clustering algorithms such as DBSCAN compared to popular objective-based approaches such as k-means [2] and spectral clustering [53] is that density-based algorithms are non-parametric. As a result, DBSCAN makes very few assumptions on the data, automatically finds the number of clusters, and allows clusters to be of arbitrary shape and size [13]. However, one of the drawbacks is that it has a worst-case quadratic runtime [16]. With the continued growth of modern datasets in both size and richness, non-parametric unsupervised procedures are becoming ever more important in understanding such datasets. Thus, there is a critical need to establish more efficient and scalable versions of these algorithms.

The computation of DBSCAN can be broken up into two steps. The first is computing the ϵ-neighborhood graph of the data points, where the ϵ-neighborhood graph is defined with data points as vertices and edges between pairs of points that are distance at most ϵ apart. The second is processing the neighborhood graph to extract the clusters. The first step has worst-case quadratic complexity simply due to the fact that each data point may have of order linear number of points in its ϵ-neighborhood for sufficiently high ϵ. However, even if the ϵ-neighborhood graph does not

---

*Equal contribution.

have such an order of edges, computing this graph remains costly: for each data point, we must query for neighbors in its ϵ-neighborhood, which is worst-case linear time for each point. There has been much work done in using space-partitioning data structures such as KD-Trees [4] to improve neighborhood queries, but these methods still run in linear time in the worst-case. Approximate methods (e.g. [23, 9]) answer queries in sub-linear time, but such methods come with few guarantees. The second step is processing the neighborhood graph to extract the clusters, which consists in finding the connected components of the subgraph induced by nodes with degree above a certain threshold (i.e. the MinPts hyperparameter in the original DBSCAN [13]). This step is linear in the number of edges in the ϵ-neighborhood graph.

Our proposal is based on a simple but powerful insight: the full ϵ-neighborhood graph may not be necessary to extract the desired clustering. We show that we can subsample the edges of the neighborhood graph while still preserving the connected components of the core-points (the high density points) on which DBSCAN's clusters are based.

To analyze this idea, we assume that the points are sampled from a distribution defined by a density function satisfying certain standard [26, 46] conditions (e.g., cluster density is sufficiently high, and clusters do not become arbitrarily thin). Such an assumption is natural because DBSCAN recovers the high-density regions as clusters [13, 26]. Under this assumption we show that the minimum cut of the ϵ-neighborhood graph is as large as $\Omega(n)$, where n is the number of datapoints. This, combined with a sampling lemma by Karger [28], implies that we can sample as little as $O(\log n/n)$ of the edges uniformly while preserving the connected components of the ϵ-neighborhood graph *exactly*. - Our algorithm, SNG-DBSCAN, proceeds by constructing and processing this subsampled ϵ-neighborhood graph all in $O(n \log n)$ time. Moreover, our procedure only requires access to $O(n \log n)$ similarity queries for random pairs of points (adding an edge between pairs if they are at most ϵ apart). Thus, unlike most implementations of DBSCAN which take advantage of space-partitioning data structures, we don't require the embeddings of the datapoints themselves. In particular, our method is compatible with arbitrary similarity functions instead of being restricted to a handful of distance metrics such as the Euclidean.

We provide an extensive empirical analysis showing that SNG-DBSCAN is effective on real datasets. We show on large datasets (on the order of a million datapoints) that we can subsample as little as 0.1% of the neighborhood graph and attain competitive performance to sci-kit learn's implementation of DBSCAN while consuming far fewer resources – as much as 200x speedup and 250x less RAM consumption on cloud machines with up to 750GB of RAM. In fact, for larger settings of ϵ on these datasets, DBSCAN fails to run at all due to insufficient RAM. We also show that our method is effective even on smaller datasets. Sampling between 1% to 30% of the edges depending on the dataset, SNG-DBSCAN shows a nice improvement in runtime while still maintaining competitive clustering performance.

2 Related Work

There is a large body of work on making DBSCAN more scalable. Due to space we can only mention some of these works here. The first approach is to more efficiently perform the nearest neighbor queries that DBSCAN uses when constructing the neighborhood graph either explicitly or implicitly [21, 31]. However, while these methods do speed up the computation of the neighborhood graph, they may not save memory costs overall as the number of edges remains similar. Our method of subsampling the neighborhood graph brings both memory and computational savings.

A natural idea for speeding up the construction of the nearest neighbors graph is to compute it approximately, for example by using locality-sensitive hashing (LSH) [24] and thus improving the overall running time [41, 57]. At the same time, since the resulting nearest neighbor graph is incomplete, the current state-of-the-art LSH-based methods lack any guarantees on the quality of the solutions they produce. This is in sharp contrast with SNG-DBSCAN, which, under certain assumptions, gives exactly the same result as DBSCAN.

Another approach is to first find a set of "leader" points that preserve the structure of the original dataset and cluster those "leader" points first. Then the remaining points are clustered to these "leader points" [17, 52]. Liu [32] modified DBSCAN by selecting clustering seeds among unlabeled core points to reduce computation time in regions that have already been clustered. Other heuristics include [37, 30]. More recently, Jang and Jiang [25] pointed out that it's not necessary to compute

the density estimates for each of the data points and presented a method that chooses a subsample of the data using k-centers to save runtime on density computations. These approaches all reduce the number of data points on which we need to perform the expensive neighborhood graph computation. SNG-DBSCAN preserves all of the data points but subsamples the edges instead.

There are also a number of approaches based on leveraging parallel computing [7, 37, 18], which includes MapReduce based approaches [15, 20, 8, 35]. Then there are also distributed approaches to DBSCAN where data is partitioned across different locations, and there may be communication cost constraints [34, 33]. Andrade et al. [1] provides a GPU implementation. In this paper, we assume a single processor, although our method can be implemented in parallel which can be a future research direction.

3 Algorithm

Algorithm 1 Subsampled Neighborhood Graph DBSCAN (SNG-DBSCAN)

Inputs: $X_{[n]}$, sampling rate s, ϵ, MinPts
Initialize graph $G = (X_{[n]}, \varnothing)$.
For each $x \in X_{[n]}$, sample $\lceil sn \rceil$ examples from $X_{[n]}, x_{i_1}, ..., x_{i_{\lceil sn \rceil}}$ and add edge (x, x_{i_j}) to G if $|x - x_{i_j}| \leqslant \epsilon$ for $j \in [[sn]]$.
Let $\mathcal{K} := \{K_1, ..., K_\ell\}$ be the connected components of the subgraph of G induced by vertices of degree at least MinPts.
Initialize $C_i = \varnothing$ for $i \in [\ell]$ and define $\mathcal{C} := \{C_1, ..., C_\ell\}$. For each $x \in X_{[n]}$, add x to C_i if $x \in K_i$. Otherwise, if x is connected to some point in K_i, add x to C_i (if multiple such $i \in [\ell]$ exist, choose one arbitrarily).
return $\mathcal{C}$.

We now introduce our algorithm SNG-DBSCAN (Algorithm 1). It proceeds by first constructing the sampled ϵ-neighborhood graph given a sampling rate s. We do this by initializing a graph whose vertices are the data points, sampling s fraction of all pairs of data points, and adding corresponding edges to the graph if points are less than ϵ apart. Compared to DBSCAN, the latter computes the full ϵ-neighborhood graph, typically using space-partitioning data structures such as kd-trees, while SNG-DBSCAN can be seen as using a sampled version of brute-force (which looks at each pair of points to compute the graph). Despite not leveraging space-partitioning data structures, we show in the experiments that SNG-DBSCAN can still be much more scalable in both runtime and memory than DBSCAN.

The remaining stage of processing the graph is the same as in DBSCAN, where we find the core-points (points with degree at least degree MinPts), compute the connected components induced by the core-points, and then cluster the border-points (those within ϵ of a connected component). The remaining points are not clustered and are referred to as noise-points or outliers. The entire procedure runs in $O(sn^2)$ time. We will show in the theory that s can be of order $O(\log n/n)$ while still maintaining statistical consistency guarantees of recovering the
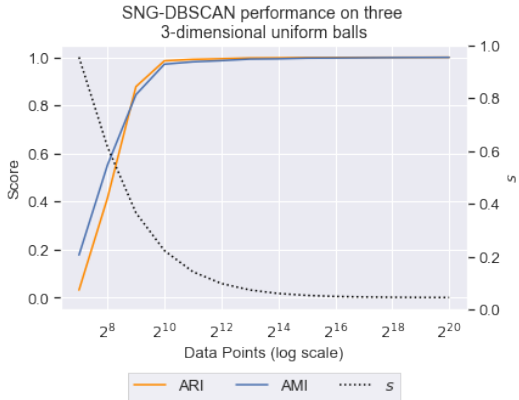

Figure 1: **SNG-DBSCAN on simulated uniform balls**. Data is generated as a uniform mixture of 3 balls in $\mathbb{R}^3$, where each ball is a *true* cluster. The x-axis is the number of examples n on a log scale. We set $s = \lceil 20 \cdot \log n/n \rceil$ and show the performance of SNG-DBSCAN under two clustering metrics Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI). We see that indeed, sampling $s \approx \log n/n$ is sufficient to recover the right clustering for sufficiently large n, as the theory suggests. Clustering quality is low at the beginning because the sample size was insufficient to learn the clusters.

3

true clusters under certain density assumptions. Under this setting, the procedure runs in $O(n \log n)$ time, and we show on simulations in Figure 1 that such a setting for s is sufficient.

4 Theoretical Analysis

For the analysis, we assume that we draw n i.i.d. samples $X_{[n]} := \{x_1, ..., x_n\}$ from a distribution $\mathcal{P}$. The goal is to show that the sampled version of DBSCAN (Algorithm 1) can recover the true clusters with statistical guarantees, where the true clusters are the connected components of a particular upper-level set of the underlying density function, as it is known that DBSCAN recovers such connected components of a level set [26, 55, 45].

We show results in two situations: (1) when the clusters are well-separated and (2) results for recovering the clusters of a particular level of the density under more general assumptions.

4.1 Recovering Well-Separated Clusters

We make the following Assumption 1 on $\mathcal{P}$. The assumption is three-fold. The first part ensures that the true clusters are of sufficiently high density level and the noise areas are of sufficiently low density. The second part ensures that the true clusters are pairwise separated by a sufficiently wide distance and that they are also separated away from the noise regions; such an assumption is common in analyses of cluster trees (e.g. [5, 6, 26]). Finally, the last part ensures that the clusters don't become arbitrarily thin anywhere. Otherwise it would be difficult to show that the entire cluster will be recovered as one connected component. This has been used in other analyses of level-set estimation [44].

Assumption 1. *Data is drawn from distribution $\mathcal{P}$ on $\mathbb{R}^D$ with density function p. There exists $R_s, R_0, \rho, \lambda_C, \lambda_N > 0$ and compact connected sets $C_1, ..., C_\ell \subset \mathbb{R}^D$ and subset $N \subset \mathbb{R}^D$ such that*

- *$p(x) \geqslant \lambda_C$ for all $x \in C_i, i \in [\ell]$ (i.e. clusters are high-density), $p(x) \leqslant \lambda_N$ for all $x \in N$ (i.e. outlier regions are low-density), $p(x) = 0$ everywhere else, and that $\rho \cdot \lambda_C > \lambda_N$ (the cluster density is sufficiently higher than noise density).*

- *$\min_{x \in C_i, x' \in C_j} |x - x'| \geqslant R_s$ for $i \neq j$ and $i, j \in [\ell]$ (i.e clusters are separated) and $\min_{x \in C_i, x' \in N} |x - x'| \geqslant R_s$ for $i \in [\ell]$ (i.e. outlier regions are away from the clusters).*

- *For all $0 < r < R_0$, $x \in C_i$ and $i \in [\ell]$ we have Volume$(B(x, r) \cap C_i) \geqslant \rho \cdot v_D \cdot r^D$ where $B(x, r) := \{x' : |x - x'| \leqslant r\}$ and v_D is the volume of a unit D-dimensional ball. (i.e. clusters don't become arbitrarily thin).*

The analysis proceeds in three steps:

1. We give a lower bound on the min-cut of the subgraph of the ϵ-neighborhood graph corresponding to each cluster. This will be useful later as it determines the sampling rate we can use while still ensure these subgraphs remain connected. (Lemma 1)

2. We use standard concentration inequalities to show that if MinPts is set appropriately, we will with high probability determine which samples belong to clusters and which ones don't in the sampled graph. (Lemma 2)

3. We then combine these two results to give a precise bound on the sampling rate s and sample complexity n to show that Algorithm 1 properly identifies the clusters. (Theorem 1)

We now give the lower bound on the min-cut of the subgraph of the ϵ-neighborhood graph corresponding to each cluster. This will be useful later as it determines the sampling rate we can use while still ensuring these subgraphs remain connected. As a reminder, for a graph $G = (V, E)$, the size of the cut-set of $S \subseteq V$ is defined as $\text{Cut}_G(S, V \backslash S) := |\{(p, q) \in E : p \in S, q \in V \backslash S\}|$ and the size of the min-cut of G is the smallest proper cut-set size: $\text{MinCut}(G) := \min_{S \subset V, S \neq \varnothing, S \neq V} \text{Cut}_G(S, V \backslash S)$.

Lemma 1 (Lower bound on min-cut of ϵ-neighborhood graph of core-points). *Suppose that Assumption 1 holds and $\epsilon < R_0$. Let $\gamma > 0$. Then there exists a constant $C_{\delta, D, p}$ depending only on D and p such that the following holds for $n \geqslant C_{D,p} \cdot \log(2/\delta) \cdot \left(\frac{1}{\epsilon}\right)^D \cdot \log^{1+\gamma}\left(\frac{1}{\epsilon}\right)$. Let $G_{n,\epsilon}$ be the*

ϵ-neighborhood graph of $X_{[n]}$. Let $G_{n,\epsilon}(i)$ be the subgraph of $G_{n,\epsilon}$ with nodes in C_i. Then with probability at least $1 - \delta$, for each $i \in [\ell]$, we have that

$$MinCut(G_{n,\epsilon}(i)) \geqslant \frac{1}{4} \cdot \lambda_C \cdot \rho \cdot v_D \cdot \epsilon^D \cdot n.$$

We next show that if MinPts is set appropriately, we will with high probability determine which samples belong to clusters and which ones don't in the sampled graph.

Lemma 2. *Suppose that Assumption 1 holds and $\epsilon < \min\{R_0, R_S\}$. Let $\delta, \gamma > 0$. There exists a universal constant C such that the following holds. Let the sampling rate be s and suppose*

$$\lambda_N \cdot v_D \cdot \epsilon^D < \frac{minPts}{sn} < \rho \cdot \lambda_C \cdot v_D \cdot \epsilon^D,$$

and $sn \geqslant \frac{C}{\Delta^2} \cdot \log^{1+\gamma}\left(\frac{1}{\Delta}\right) \cdot \log(1/\delta)$, where $\Delta := \min\{\frac{minPts}{sn} - \lambda_N \cdot v_D \cdot \epsilon^D, \rho \cdot \lambda_C \cdot v_D \cdot \epsilon^D - \frac{minPts}{sn}\}$. Then with probability at least $1 - \delta$, all samples in C_i for some $i \in [\ell]$ are identified as core-points and the rest are noise points.

The next result shows a rate at which we can sample a graph while still have it be connected, which depends on the min-cut and the size of the graph. It follows from classical results in graph theory that cut sizes remain preserved under sampling [28].

Lemma 3. *There exists universal constant C such that the following holds. Let G be a graph with min-cut m and $0 < \delta < 1$. If*

$$s \geqslant \frac{C \cdot (\log(1/\delta) + \log(n))}{m},$$

then with probability at least $1 - \delta$, the graph G_s obtained by sampling each edge of G with probability s is connected.

We now give the final result, which follows from combining Lemmas 1, 2, and 3.

Theorem 1. *Suppose that Assumption 1 holds and $\epsilon < \min\{R_0, R_S\}$. Let $\delta, \gamma > 0$. There exist universal constants C_1, C_2 and constant $C_{D,p}$ depending only on D and p such that the following holds. Suppose*

$$\lambda_N \cdot v_D \cdot \epsilon^D < \frac{minPts}{sn} < \rho \cdot \lambda_C \cdot v_D \cdot \epsilon^D, \qquad s \geqslant \frac{C_1 \cdot (\log(1/\delta) + \log n)}{\lambda_C \cdot \rho \cdot v_D \cdot \epsilon^D \cdot n},$$

and

$$n \geqslant \max\left\{ C_{D,p} \cdot \log(2/\delta) \cdot \left(\frac{1}{\epsilon}\right)^D \cdot \log^{1+\gamma}\left(\frac{1}{\epsilon}\right), \frac{1}{s}\frac{C_2}{\Delta^2} \cdot \log^{1+\gamma}\left(\frac{1}{\Delta}\right) \cdot \log(1/\delta) \right\},$$

where $\Delta := \min\{\frac{minPts}{sn} - \lambda_N \cdot v_D \cdot \epsilon^D, \rho \cdot \lambda_C \cdot v_D \cdot \epsilon^D - \frac{minPts}{sn}\}$.

Then with probability at least $1 - 3\delta$, Algorithm 1 returns (up to permutation) the clusters $\{C_1 \cap X_{[n]}, C_2 \cap X_{[n]}, ..., C_\ell \cap X_{[n]}\}$.

Remark 1. *As a consequence, we can take $s = \Theta(\log n / \epsilon^D n)$, and, for n sufficiently large, the clusters are recovered with high probability, leading to a computational complexity of $O(\frac{1}{\epsilon^D} \cdot n \log n)$ for Algorithm 1.*

4.2 Recovering Clusters of a Particular Level of the Density

We show level-set estimation rates for estimating a particular level λ (i.e. $L_f(\lambda) := \{x \in \mathcal{X} : f(x) \geqslant \lambda\}$) given that hyperparameters of SNG-DBSCAN are set appropriately depending on density f, s, λ and n under the following assumption which characterizes the regularity of the density function around the boundaries of the level-set. This assumption is standard amongst analyses in level-set estimation e.g. [26, 44].

Assumption 2. *f is a uniformly continuous density on compact set $\mathcal{X} \subseteq \mathbb{R}^D$. There exists $\beta, \check{C}, \hat{C}, r_c > 0$ such that the following holds for all $x \in B(L_f(\lambda), r_c) \backslash L_f(\lambda)$: $\check{C} \cdot d(x, L_f(\lambda))^\beta \leqslant \lambda - f(x) \leqslant \hat{C} \cdot d(x, L_f(\lambda))^\beta$, where $d(x, A) := \inf_{x' \in A} |x - x'|$ and $B(A, r) := \{x : d(x, A) \leqslant r\}$.*

5

Theorem 2. *Suppose that Assumption 2 holds, $0 < \delta < 1$, and ϵ and MinPts satisfies the following:*

$$\epsilon^\beta = \frac{1}{\widehat{C}} \left(\lambda - \frac{MinPts}{v_D \cdot \epsilon^D \cdot s \cdot n} - \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}} \right).$$

Let $\widehat{L_f(\lambda)}$ be the union of all the clusters returned by SNG-DBSCAN. Then, for n sufficiently large depending on $f, \epsilon, MinPts$, the following holds with probability at least $1 - \delta$:

$$d_{Haus}(\widehat{L_f(\lambda)}, L_f(\lambda)) \leqslant C \cdot \left(\left(\frac{\log(4n) + \log(1/\delta)}{s \cdot n} \right)^{1/2\beta} + \epsilon \right),$$

where C is some constant depending on f and d_{Haus} is Hausdorff distance.

Given these results, they can be extended them to obtain clustering results with the same convergence rates (i.e. showing that SNG-DBSCAN recovers the connected components of the level-set individually) in a similar way as shown in the previous result.

5 Experiments

Datasets and hyperparameter settings: We compare the performance of SNG-DBSCAN against DBSCAN on 5 large (~1,000,000+ datapoints) and 12 smaller (~100 to ~100,000 datapoints) datasets from UCI [11] and OpenML [50]. Details about each large dataset shown in the main text are summarized in Figure 2, and the rest, including all hyperparameter settings, can be found in the Appendix. Due to space constraints, we couldn't show the results for all of the datasets in the main text. The settings of MinPts and range of ϵ that we ran SNG-DBSCAN and DBSCAN on, as well as how s was chosen, are shown in the Appendix. For simplicity we fixed MinPts and only tuned ϵ which is the more essential hyperparameter. We compare our implementation of SNG-DBSCAN to that of sci-kit learn's DBSCAN [39], both of which are implemented with Cython, which allows the code to have a Python API while the expensive computations are done in a C/C++ backend.

Clustering evaluation: To score cluster quality, we use two popular clustering scores: the Adjusted Rand Index (ARI) [22] and Adjusted Mutual Information (AMI) [51] scores. ARI is a measure of the fraction of pairs of points that are correctly clustered to the same or different clusters. AMI is a measure of the cross-entropy of the clusters and the ground truth. Both are normalized and adjusted for chance, so that the perfect clustering receives a score of 1 and a random one receives a score of 0. The datasets we use are classification datasets where we cluster on the features and use the labels to evaluate our algorithm's clustering performance. It is standard practice to evaluate performance using the labels as a proxy for the ground truth [27, 25].

5.1 Performance on Large Datasets

We now discuss the performance on the large datasets, which we ran on a cloud compute environment. Due to computational costs, we only ran DBSCAN once for each ϵ setting. We ran SNG-DBSCAN 10 times for each ϵ setting and averaged the clustering scores and runtimes. The results are summarized in Figure 3, which reports the highest clustering scores for the respective algorithms along with the runtime and RAM usage required to achieve these scores. We also plot the clustering performance and runtime/RAM usage across different settings of ϵ in Figure 4.

	n	D	c	s	mPts
Australian	1,000,000	14	2	0.001	10
Still [47]	949,983	3	6	0.001	10
Watch [47]	3,205,431	3	7	0.01	10
Satimage	1,000,000	36	6	0.02	10
Phone [47]	1,000,000	3	7	0.001	10

Figure 2: *Summary of larger datasets and hyperparameter settings used.* Includes dataset size (n), number of features (D), number of clusters (c), and the fraction s of neighborhood edges kept by SNG-DBSCAN. Datasets Phone, Watch, and Still come from the Heterogeneity Activity Recognition dataset. Phone originally had over 13M points, but in order to feasibly run DBSCAN, we used a random 1M sample. For the full chart, including settings for ϵ, see the Appendix.

6

	Base ARI	SNG ARI	Base AMI	SNG AMI
Australian	**0.0933** (6h 4m) 282 GB	**0.0933** (1m 36s) 1.5 GB	**0.1168** (4h 28m) 146 GB	0.1167 (1m 36s) 1.5 GB
Still	0.7901 (14m 37s) 419 GB	**0.7902** (1m 21s) 1.6 GB	0.8356 (14m 37s) 419 GB	**0.8368** (1m 57s) 1.5 GB
Watch	0.1360 (27m 2s) 518 GB	**0.1400** (1h 27m) 8.7 GB	0.1755 (7m 35s) 139 GB	**0.1851** (1h 29m) 7.6 GB
Satimage	0.0975 (1d 3h) 11 GB	**0.0981** (33m 22s) 2.1 GB	0.1019 (1d 3h) 11 GB	**0.1058** (33m 22s) 2.1 GB
Phone	0.1902 (12m 36s) 138 GB	**0.1923** (59s) 1.7 GB	0.2271 (4m 48s) 32 GB	**0.2344** (46s) 1.5 GB

Figure 3: *Performance on large datasets.* Best Adjusted Rand Index and Adjusted Mutual Information Scores for both DBSCAN and SNG-DBSCAN after ϵ-tuning, shown with their associated runtimes and RAM used. Highest scores for each dataset are bolded. We see that SNG-DBSCAN is competitive against DBSCAN in terms of clustering quality while using much fewer resources. On Australian, we see an over 200x speedup. On Still, we see an over 250x reduction in RAM.
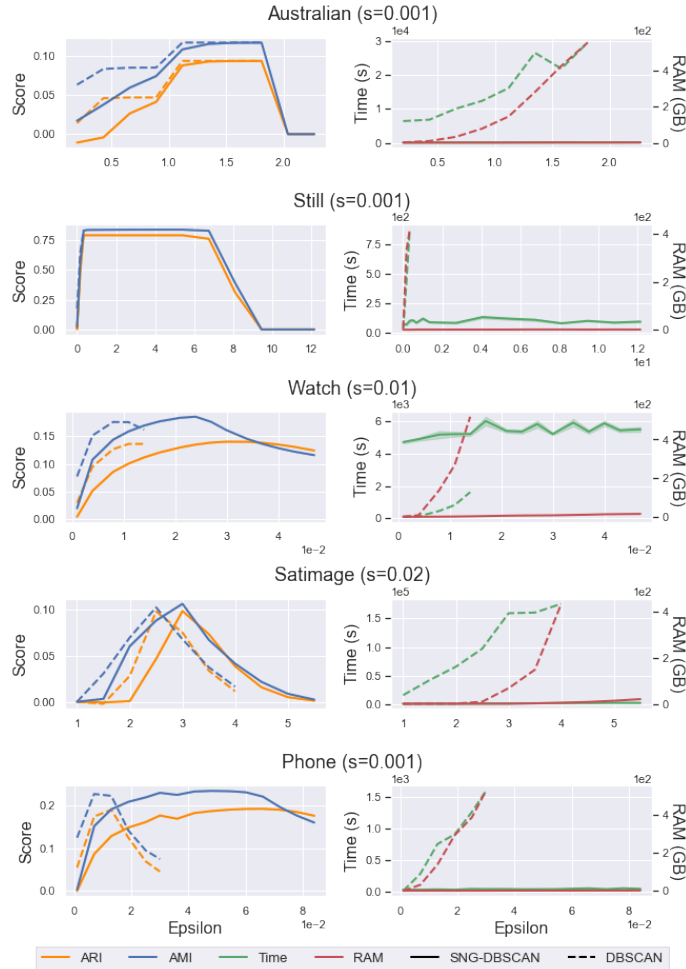

Figure 4: **Large dataset results across** ϵ. We show the performance of SNG-DBSCAN and DB-SCAN on five large datasets. SNG-DBSCAN values are averaged over 10 runs, and 95% confidence intervals (standard errors) are shown. We ran these experiments on a cloud environment and plot the adjusted RAND index score, adjusted mutual information score, runtime, and RAM used across a wide range of ϵ. DBSCAN was run with MinPts $= 10$, and SNG-DBSCAN was run with MinPts $= \max(2, \lfloor 10 \cdot s \rfloor)$ for all datasets. No data is given for DBSCAN for ϵ settings requiring more than 750GB of RAM as it wasn't possible for DBSCAN to run on these machines.

7

From Figure 4, we see that DBSCAN often exceeds the 750GB RAM limit on our machines and fails to complete; the amount of memory (as well as runtime) required for DBSCAN escalates quickly as ϵ increases, suggesting that the size of the ϵ-neighborhood graph grows quickly. Meanwhile, SNG-DBSCAN's memory usage remains reasonable and increases slowly across ϵ. This suggests that SNG-DBSCAN can run on much larger datasets infeasible for DBSCAN, opening the possibilities for applications which may have previously not been possible due to scalability constraints – all the while attaining competitive clustering quality (Figure 3).

Similarly, SNG-DBSCAN shows a significant runtime improvement for almost all datasets and stays relatively constant across epsilon. We note that occasionally (e.g. Watch dataset for small ϵ), SNG-DBSCAN is slower. This is likely because SNG-DBSCAN does not take advantage of space-partitioning data structures such as kd-trees. This is more likely on lower dimensional datasets where space-partitioning data structures tend to perform faster as the number of partitions is exponential in dimension [4]. Conversely, we see the largest speedup on Australian, which is the highest dimension large dataset we evaluated. However, DBSCAN was unable to finish clustering Watch past the first few values of ϵ due to exceeding memory limits. During preliminary experimentation, but not shown in the results, we tested DBSCAN on the UCI Character Font Images dataset which has dimension 411 and 745,000 datapoints. DBSCAN failed to finish after running on the cloud machine for over 6 days. Meanwhile, SNG-DBSCAN was able to finish with the same ϵ settings in under 15 mins using the $s = 0.01$ setting. We didn't show the results here because we were unable to obtain clustering scores for DBSCAN.

5.2 Performance on smaller datasets

We now show that we don't require large datasets to enjoy the advantages of SNG-DBSCAN. Speedups are attainable for even the smallest datasets without sacrificing clustering quality. Due to space constraints, we provide the summary of the datasets and hyperparameter setting details in the Appendix. In Figure 5, we show performance metrics for some datasets under optimal tuning. The results for the rest of the datasets are in the Appendix, where we also provide charts showing performance across ϵ and different settings of s to better understand the effect of the sampling rate on cluster quality– there we show that SNG-DBSCAN is stable in the s hyperparameter. Overall, we see that SNG-DBSCAN can give considerable savings in computational costs while remaining competitive in clustering quality on smaller datasets.

	Base ARI	SNG ARI	Base AMI	SNG AMI
Wearable	0.2626 (55s)	**0.3064** (8.2s)	**0.4788** (26s)	0.4720 (6.6s)
Iris	**0.5681** (<0.01s)	**0.5681** (<0.01s)	**0.7316** (<0.01s)	**0.7316** (<0.01s)
LIBRAS	0.0713 (0.03s)	**0.0903** (<0.01s)	0.2711 (0.02s)	**0.3178** (<0.01s)
Page Blocks	0.1118 (0.38s)	**0.1134** (0.06s)	**0.0742** (0.49s)	0.0739 (0.06s)
kc2	0.3729 (<0.01s)	**0.3733** (<0.01s)	**0.1772** (<0.01s)	0.1671 (<0.01s)
Faces	0.0345 (1.7s)	**0.0409** (0.16s)	0.2399 (1.7s)	**0.2781** (0.15s)
Ozone	0.0391 (<0.01s)	**0.0494** (<0.01s)	0.1214 (<0.01s)	**0.1278** (<0.01s)
Bank	0.1948 (3.4s)	**0.2265** (0.03s)	0.0721 (3.3s)	**0.0858** (0.03s)

Figure 5: *Clustering performance.* Best Adjusted Rand Index and Adjusted Mutual Information scores for both DBSCAN and SNG-DBSCAN under optimal tuning are given with associated runtimes in parentheses. Highest scores for each dataset are bolded. Full table can be found in the Appendix.

5.3 Performance against DBSCAN++

We now compare the performance of SNG-DBSCAN against a recent DBSCAN speedup called DBSCAN++[25], which proceeds by performing k-centers to select $\lfloor sn \rfloor$ candidate points, computing densities for these candidate points and using these densities to identify a subset of core-points, and finally clustering the dataset based on the ϵ-neighborhood graph of these core-points. Like our method, DBSCAN++ also has $O(sn^2)$ runtime. SNG-DBSCAN can be seen as subsampling edges while DBSCAN++ subsamples the vertices.

We show comparative results on our large datasets in Figure 6. We ran DBSCAN++ with the same sampling rate s as SNG-DBSCAN. We see that SNG-DBSCAN is competitive and beats DBSCAN++ on 70% of the metrics. Occasionally, DBSCAN++ fails to produce reasonable clusters with the s given, as shown by the low scores for some datasets. We see that SNG-DBSCAN is slower than DBSCAN++ for the low-dimensional datasets (i.e. Phone, Watch, and Still, which are all of dimension 3) while there is a considerable speedup on Australian. Like in the experiments on the large datasets against DBSCAN, this is due to the fact that DBSCAN++, like DBSCAN, leverages space-partitioning data structures such as kd-trees, which are faster in low dimensions. Overall, we see that SNG-DBSCAN is a better choice over DBSCAN++ when using the same sampling rate.

	DBSCAN++ ARI	SNG ARI	DBSCAN++ AMI	SNG AMI
Australian	**0.1190** (1185s)	0.0933 (96s)	0.1166 (1178s)	**0.1167** (96s)
	1.6 GB	1.5 GB	1.6 GB	1.5 GB
Satimage	0.0176 (5891s)	**0.0981** (2002s)	**0.1699** (4842s)	0.1058 (2002s)
	2.3 GB	2.1 GB	2.3 GB	2.1 GB
Phone	0.0001 (11s)	**0.1923** (59s)	0.0023 (11s)	**0.2344** (46s)
	1.0 GB	1.7 GB	1.0 GB	1.5 GB
Watch	0.0000 (973s)	**0.1400** (5230s)	0.0025 (1017s)	**0.1851** (5371s)
	1.4 GB	8.7 GB	1.4 GB	7.6 GB
Still	0.7900 (12s)	**0.7902** (81s)	**0.8370** (15s)	0.8368 (117s)
	1.4 GB	1.6 GB	1.4 GB	1.5 GB

Figure 6: *SNG-DBSCAN against DBSCAN++[25] on large datasets tuned over ϵ.*

	DBSCAN++ ARI	SNG ARI	DBSCAN++ AMI	SNG AMI
Wearable	0.2517	**0.3263**	0.4366	**0.5003**
Iris	**0.6844**	0.5687	**0.7391**	0.7316
LIBRAS	**0.1747**	0.0904	**0.3709**	0.2985
Page Blocks	0.0727	**0.1137**	0.0586	**0.0760**
kc2	0.3621	**0.3747**	0.1780	**0.1792**
Faces	**0.0441**	0.0424	**0.2912**	0.2749
Ozone	**0.0627**	0.0552	0.1065	**0.1444**
Bank	**0.2599**	0.2245	0.0874	**0.0875**
Ionosphere	0.1986	**0.6359**	0.2153	**0.5615**
Mozilla	0.1213	**0.2791**	0.1589	**0.1806**
Tokyo	0.4180	**0.4467**	0.2793	**0.3147**
Vehicle	**0.1071**	0.0971	**0.1837**	0.1807

Figure 7: *Performance comparison of SNG-DBSCAN and DBSCAN++ (with uniform sampling) with ϵ tuned appropriately and sampling rate tuned over grid [0.1,0.2,..,0.9] to maximize ARI and AMI clustering scores.*

In Figure 7, we compare DBSCAN++ and SNG-DBSCAN on the smaller datasets under optimal tuning of the sampling rate as well as the other hyperparameters. We find that under optimal tuning, these algorithms have competitive performance against each other.

6   Conclusion

Density clustering has had a profound impact on machine learning and data mining; however, it can be very expensive to run on large datasets. We showed that the simple idea of subsampling the neighborhood graph leads to a procedure, which we call SNG-DBSCAN that runs in $O(n \log n)$ time and comes with statistical consistency guarantees. We showed empirically on a variety of real datasets that SNG-DBSCAN can offer a tremendous savings in computational resources while maintaining clustering quality. Future research directions include using adaptive instead of uniform sampling of edges, combining SNG-DBSCAN with other speedup techniques, and paralellizing the procedure for even faster computation.

Broader Impact

As stated in the introduction, DBSCAN has a wide range of applications within machine learning and data mining. Our contribution is a more efficient variant of DBSCAN. The potential impact of SNG-DBSCAN lies in considerable savings in computational resources and further applications of density clustering which weren't possible before due to scalability constraints.

Acknowledgments and Disclosure of Funding

References

[1] Guilherme Andrade, Gabriel Ramos, Daniel Madeira, Rafael Sachetto, Renato Ferreira, and Leonardo Rocha. G-dbscan: A gpu accelerated algorithm for density-based clustering. *Procedia Computer Science*, 18:369–378, 2013.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[3] Fabio Baselice, Luigi Coppolino, Salvatore D'Antonio, Giampaolo Ferraioli, and Luigi Sgaglione. A dbscan based approach for jointly segment and classify brain mr images. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2993–2996. IEEE, 2015.

[4] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] Kamalika Chaudhuri and Sanjoy Dasgupta. Rates of convergence for the cluster tree. In *Advances in Neural Information Processing Systems*, pages 343–351, 2010.

[6] Kamalika Chaudhuri, Sanjoy Dasgupta, Samory Kpotufe, and Ulrike von Luxburg. Consistent procedures for cluster tree estimation and pruning. *IEEE Transactions on Information Theory*, 60(12):7900–7912, 2014.

[7] Min Chen, Xuedong Gao, and HuiFei Li. Parallel dbscan with priority r-tree. In *Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on*, pages 508–511. IEEE, 2010.

[8] Bi-Ru Dai and I-Chang Lin. Efficient map/reduce-based dbscan algorithm with optimized data partition. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 59–66. IEEE, 2012.

[9] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[10] Ahmet Can Diker and Elvin Nasibov. Estimation of traffic congestion level via fn-dbscan algorithm by using gps data. In *2012 IV International Conference" Problems of Cybernetics and Informatics"(PCI)*, pages 1–4. IEEE, 2012.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[12] Hossein Saeedi Emadi and Sayyed Majid Mazinani. A novel anomaly detection algorithm using dbscan and svm in wireless sensor networks. *Wireless Personal Communications*, 98(2): 2025–2035, 2018.

[13] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[14] Tianhui Fan, Naijing Guo, and Yujie Ren. Consumer clusters detection with geo-tagged social network data using dbscan algorithm: a case study of the pearl river delta in china. *GeoJournal*, pages 1–21, 2019.

[15] Yan Xiang Fu, Wei Zhong Zhao, and Hui Fang Ma. Research on parallel dbscan algorithm design based on mapreduce. In *Advanced Materials Research*, volume 301, pages 1133–1138. Trans Tech Publ, 2011.

[16] Junhao Gan and Yufei Tao. Dbscan revisited: mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 519–530. ACM, 2015.

[17] ZHOU Shui Geng, ZHOU Ao Ying, CAO Jing, and HU Yun Fa. A fast density based clustering algorithm [j]. *Journal of Computer Research and Development*, 11:001, 2000.

[18] Markus Götz, Christian Bodenstein, and Morris Riedel. Hpdbscan: highly parallel dbscan. In *Proceedings of the workshop on machine learning in high-performance computing environments*, page 2. ACM, 2015.

[19] Chao-Hua Guan, Yong-Dan Chen, Hui-Yan Chen, and Jian-Wei Gong. Improved dbscan clustering algorithm based vehicle detection using a vehicle-mounted laser scanner. *Transactions of Beijing Institute of Technology*, 30(6):732–736, 2010.

[20] Yaobin He, Haoyu Tan, Wuman Luo, Huajian Mao, Di Ma, Shengzhong Feng, and Jianping Fan. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, pages 473–480. IEEE, 2011.

[21] Ming Huang and Fuling Bian. A grid and density based fast spatial clustering algorithm. In *Artificial Intelligence and Computational Intelligence, 2009. AICI'09. International Conference on*, volume 4, pages 260–263. IEEE, 2009.

[22] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1): 193–218, 1985.

[23] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[24] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 618–625, 1997.

[25] Jennifer Jang and Heinrich Jiang. Dbscan++: Towards fast and scalable density clustering. In *International Conference on Machine Learning*, pages 3019–3029, 2019.

[26] Heinrich Jiang. Density level set estimation on manifolds with dbscan. In *International Conference on Machine Learning*, pages 1684–1693, 2017.

[27] Heinrich Jiang, Jennifer Jang, and Samory K Kpotufe. Quickshift++: Provably good initializations for sample-based mean shift. In *35th International Conference on Machine Learning, ICML 2018*, pages 3591–3600. International Machine Learning Society (IMLS), 2018.

[28] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[29] Mehjabin Khatoon and W Aisha Banu. An efficient method to detect communities in social networks using dbscan algorithm. *Social Network Analysis and Mining*, 9(1):9, 2019.

[30] Marzena Kryszkiewicz and Piotr Lasek. Ti-dbscan: Clustering with dbscan by means of the triangle inequality. In *International Conference on Rough Sets and Current Trends in Computing*, pages 60–69. Springer, 2010.

[31] K Mahesh Kumar and A Rama Mohan Reddy. A fast dbscan clustering algorithm by accelerating neighbor searching using groups method. *Pattern Recognition*, 58:39–48, 2016.

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

[33] Alessandro Lulli, Matteo Dell'Amico, Pietro Michiardi, and Laura Ricci. Ng-dbscan: scalable density-based clustering for arbitrary data. *Proceedings of the VLDB Endowment*, 10(3): 157–168, 2016.

[34] Antonio Cavalcante Araujo Neto, Ticiana Linhares Coelho da Silva, Victor Aguiar Evangelista de Farias, José Antonio F Macêdo, and Javam de Castro Machado. G2p: A partitioning

approach for processing dbscan with mapreduce. In *International Symposium on Web and Wireless Geographical Information Systems*, pages 191–202. Springer, 2015.

[35] Maitry Noticewala and Dinesh Vaghela. Mr-idbscan: Efficient parallel incremental dbscan algorithm using mapreduce. *International Journal of Computer Applications*, 93(4), 2014.

[36] Ghazaleh Panahandeh and Niklas Åkerblom. Clustering driving destinations using a modified dbscan algorithm with locally-defined map-based thresholds. In *European Congress on Computational Methods in Applied Sciences and Engineering*, pages 97–103. Springer, 2015.

[37] Mostofa Ali Patwary, Diana Palsetia, Ankit Agrawal, Wei-keng Liao, Fredrik Manne, and Alok Choudhary. A new scalable parallel dbscan algorithm using the disjoint-set data structure. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 62. IEEE Computer Society Press, 2012.

[38] Michalis Pavlis, Les Dolega, and Alex Singleton. A modified dbscan clustering method to estimate retail center extent. *Geographical Analysis*, 50(2):141–161, 2018.

[39] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[40] Jianbing Shen, Xiaopeng Hao, Zhiyuan Liang, Yu Liu, Wenguan Wang, and Ling Shao. Real-time superpixel segmentation by dbscan clustering algorithm. *IEEE transactions on image processing*, 25(12):5933–5942, 2016.

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

[42] J Sayyad Shirabad and T Menzies. The promise repository of software engineering databases. school of information technology and engineering, university of ottawa, canada, 2005.

[43] J Paul Siebert. Vehicle recognition using rule based methods. 1987.

[44] Aarti Singh, Clayton Scott, Robert Nowak, et al. Adaptive hausdorff estimation of density level sets. *The Annals of Statistics*, 37(5B):2760–2782, 2009.

[45] Bharath Sriperumbudur and Ingo Steinwart. Consistency and rates for clustering with dbscan. In *Artificial Intelligence and Statistics*, pages 1090–1098, 2012.

[46] Ingo Steinwart. Adaptive density level set clustering. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 703–738, 2011.

[47] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 127–140, 2015.

[48] Thanh N Tran, Thanh T Nguyen, Tofan A Willemsz, Gijs van Kessel, Henderik W Frijlink, and Kees van der Voort Maarschalk. A density-based segmentation for 3d images, an application for x-ray micro-tomography. *Analytica chimica acta*, 725:14–21, 2012.

[49] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiú, and Hugo Fuks. Wearable computing: Accelerometers' data classification of body postures and movements. In *Brazilian Symposium on Artificial Intelligence*, pages 52–61. Springer, 2012.

[50] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

[51] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[52] P Viswanath and V Suresh Babu. Rough-dbscan: A fast hybrid density based clustering method for large data sets. *Pattern Recognition Letters*, 30(16):1477–1488, 2009.

[53] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[54] Thomas Wagner, Reinhard Feger, and Andreas Stelzer. Modification of dbscan and application to range/doppler/doa measurements for pedestrian recognition with an automotive radar system. In *2015 European Radar Conference (EuRAD)*, pages 269–272. IEEE, 2015.

[55] Daren Wang, Xinyang Lu, and Alessandro Rinaldo. Optimal rates for cluster tree estimation using kernel density estimators. *arXiv preprint arXiv:1706.03113*, 2017.

[56] Kai Wang, Xing Yu, Qingyu Xiong, Qiwu Zhu, Wang Lu, Ya Huang, and Linyu Zhao. Learning to improve wlan indoor positioning accuracy based on dbscan-krf algorithm from rss fingerprint data. *IEEE Access*, 7:72308–72315, 2019.

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

[58] Kai Yao, Dimitris Papadias, and Spiridon Bakiras. Density-based community detection in geo-social networks. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*, pages 110–119, 2019.

A Proofs

Let $\mathcal{P}_n$ be the empirical distribution w.r.t. $X_{[n]}$. We need the following result giving uniform guarantees on the masses of empirical balls with respect to the mass of true balls w.r.t. $\mathcal{P}$.

Lemma 4 (Chaudhuri and Dasgupta [5]). *Pick $0 < \delta < 1$. Then with probability at least $1 - \delta$, for every ball $B \subset \mathbb{R}^D$ and $k \geqslant D \log n$, we have*

$$\mathcal{P}(B) \geqslant C_{\delta,n} \frac{\sqrt{D \log n}}{n} \Rightarrow \mathcal{P}_n(B) > 0$$

$$\mathcal{P}(B) \geqslant \frac{k}{n} + C_{\delta,n} \frac{\sqrt{k}}{n} \Rightarrow \mathcal{P}_n(B) \geqslant \frac{k}{n}$$

$$\mathcal{P}(B) \leqslant \frac{k}{n} - C_{\delta,n} \frac{\sqrt{k}}{n} \Rightarrow \mathcal{P}_n(B) < \frac{k}{n},$$

where $C_{\delta,n} := 16 \log(2/\delta)\sqrt{D \log n}$.

Proof of Lemma 1. Suppose that S is a proper subset of the nodes in $G_{n,\epsilon}(i)$. Let $S^C = (X_{[n]} \cap C_i) \backslash S$ be the complement of S in $G_{n,\epsilon}(i)$. We lower bound the cut of S and S^C in $G_{n,\epsilon}(i)$.

Let $r_0 := \left(\frac{16 \log(2/\delta) \cdot D \log n}{\lambda_C \cdot \rho \cdot n} \right)^{1/D}$. Then for any $x \in C_i$, we have:

$$\mathcal{P}(B(x, r_0)) \geqslant \lambda_C \cdot \text{Volume}(B(x, r_0) \cap C_i) \geqslant \lambda_C \cdot \rho \cdot v_D \cdot r_0^D = C_{\delta,n} \frac{\sqrt{D \log n}}{n}.$$

Thus, by Lemma 4, there exists a sample point in $B(x, r_0)$. It follows that $\{B(x, r_0) : x \in S \cup S^C\}$ forms a cover of C_i (i.e $C_i \subseteq \bigcup \{B(x, r_0) : x \in S \cup S^C\}$). Since C_i is connected, then there exists $x \in S$ and $x' \in S^C$ such that $B(x, r_0)$ and $B(x', r_0)$ intersect and thus $|x - x'| \leqslant 2 \cdot r_0$.

The cut of S and S^C in $G_{n,\epsilon}(i)$ thus contains all edges from x to S^C and from x' to S, which is at least the number of nodes in $B(x, \epsilon) \cap B(x', \epsilon)$. We have

$$\text{Cut}(S, S^C) \geqslant |B(x, \epsilon) \cap B(x', \epsilon) \cap X_{[n]}| \geqslant |B(x, \epsilon - 2r_0) \cap X_{[n]}| = n \cdot \mathcal{P}_n(B(x, \epsilon - 2r_0)).$$

We have

$$\mathcal{P}(B(x, \epsilon - 2r_0)) \geqslant \lambda_C \cdot \rho \cdot v_D \cdot (\epsilon - 2r_0)^D \geqslant \lambda_C \cdot \rho \cdot v_D \cdot \epsilon^D \cdot \left(1 - \frac{2Dr_0}{\epsilon} \right) \geqslant \frac{1}{2} \lambda_C \cdot \rho \cdot v_D \cdot \epsilon^D,$$

which holds for n sufficiently large as in the statement of the Lemma for some $C_{D,p}$. By Lemma 4, we have

$$\mathcal{P}_n(B(x, \epsilon - 2r_0)) \geqslant \frac{1}{4} \cdot \lambda_C \cdot \rho \cdot v_D \cdot \epsilon^D,$$

which also holds for n sufficiently large as in the statement of the Lemma for some $C_{D,p}$. The result follows. $\square$

Proof of Lemma 2. Let $N_{n,s}(x, \epsilon)$ be the neighbors of x in the sampled ϵ-neighborhood graph. Suppose that $x \in C_i$ for some $i \in [\ell]$. Then by Hoeffding's inequality, we have

$$P\left(\frac{|N_{n,s}(x, \epsilon)|}{sn} \leqslant \mathcal{P}(B(x, \epsilon)) - \sqrt{\frac{\log(n) + \log(1/\delta)}{2sn}} \right) \leqslant \frac{\delta}{n}.$$

Thus, with probability at least $1 - \delta/n$, we have

$$|N_{n,s}(x, \epsilon)| \geqslant sn \cdot \mathcal{P}(B(x, \epsilon)) - \sqrt{\frac{1}{2} (\log(n) + \log(1/\delta)) \cdot sn}$$

$$\geqslant sn\rho \cdot v_D \cdot \epsilon^D \cdot \lambda_C - \sqrt{\frac{1}{2} (\log(n) + \log(1/\delta)) \cdot sn}.$$

Now, suppose that $x \in N$. Then by Hoeffding's inequality, we have

$$P\left(\frac{|N_{n,s}(x,\epsilon)|}{sn} \geqslant \mathcal{P}(B(x,\epsilon)) + \sqrt{\frac{\log(n) + \log(1/\delta)}{2sn}}\right) \leqslant \frac{\delta}{n}.$$

Thus, with probability at least $1 - \delta/n$, we have

$$|N_{n,s}(x,\epsilon)| \leqslant sn \cdot \mathcal{P}(B(x,\epsilon)) + \sqrt{\frac{1}{2}\left(\log(n) + \log(1/\delta)\right) \cdot sn}$$

$$\leqslant sn \cdot v_D \cdot \epsilon^D \cdot \lambda_N + \sqrt{\frac{1}{2}\left(\log(n) + \log(1/\delta)\right) \cdot sn}.$$

Hence, the following holds uniformly with probability at least $1 - \delta$. When $x \in C_i$ for some i, we have

$$\frac{|N_{n,s}(x,\epsilon)|}{sn} \geqslant \rho \cdot \lambda_C \cdot v_D \cdot \epsilon^D - \sqrt{\frac{\log(n) + \log(1/\delta)}{2 \cdot sn}},$$

and otherwise we have

$$\frac{|N_{n,s}(x,\epsilon)|}{sn} \leqslant \lambda_N \cdot v_D \cdot \epsilon^D + \sqrt{\frac{\log(n) + \log(1/\delta)}{2 \cdot sn}}.$$

The result follows because minPts is the threshold on $\frac{|N_{n,s}(x,\epsilon)|}{sn}$ that decides whether a point is a core-point. There are no border points because $\epsilon < R_S$ and there are no points within R_S of C_i for $i \in [\ell]$. $\qquad\square$

Proof of Lemma 3. Follows from Theorem 2.1 of [28]. $\qquad\square$

Proof of Theorem 2. There are two quantities to bound: (i) $\max_{x \in \widehat{L_f(\lambda)}} d(x, L_f(\lambda))$, and (ii) $\sup_{x \in L_f(\lambda)} d(x, \widehat{L_f(\lambda)})$. We start with (i). Define

$$\widehat{f}(x) := \frac{1}{v_D \cdot \epsilon^D} \cdot \frac{\deg_G(x)}{s \cdot n}, \qquad f_\epsilon(x) := \frac{1}{v_D \cdot \epsilon^D} \cdot \int_{x' \in B(x,\epsilon)} f(x')dx',$$

where $\deg_G$ denotes the degree of a node in the subsampled ϵ-neighborhood graph G as per Algorithm 1 and $f_\epsilon(x)$ can be viewed as the density f smoothed with a uniform kernel with radius ϵ. Then, we have for all $x \in X_{[n]}$, the probability that an edge incident to x appearing in G has probability $v_D \cdot \epsilon^D \cdot f_\epsilon(x)$. Thus we have by Hoeffding's inequality:

$$\mathbb{P}(|\widehat{f}(x) - f_\epsilon(x)| \geqslant \eta) \leqslant 1 - 2\exp(-\eta^2 \cdot s \cdot n).$$

Setting

$$\eta = \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}},$$

we have that by union bound, with probability at least $1 - \delta/2$ uniformly in $x \in X_{[n]}$:

$$|\widehat{f}(x) - f_\epsilon(x)| \leqslant \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}}.$$

Now, for $r > 0$ to be specified later, we have that if $x \in X_{[n]}$ such that $d(x, L_f(\lambda)) \geqslant r$, then by Assumption 2 for n sufficiently large depending on f, ϵ, and MinPts:

$$f(x) \leqslant \lambda - \check{C}r^\beta \Rightarrow f_\epsilon(x) \leqslant \lambda - \check{C}(r - \epsilon)^\beta.$$

Therefore,

$$\widehat{f}(x) \leqslant \lambda - \check{C}(r - \epsilon)^\beta + \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}}.$$

Hence, if the following holds:

$$\lambda - \check{C}(r - \epsilon)^\beta + \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}} < \frac{\text{MinPts}}{v_D \cdot \epsilon^D \cdot s \cdot n},$$

then x is not a core point and hence $\max_{x \in \widehat{L_f(\lambda)}} d(x, L_f(\lambda)) \leqslant r$. This holds by choosing r to be the bound specified in the theorem statement for some C depending on $\check{C}, \widehat{C}, \beta$ by generalized mean inequality. Now we move onto the other direction. We first show that each $x \in L_f(\lambda) \cap X_{[n]}$ is a core point. Indeed we have

$$\widehat{f}(x) \geqslant f_\epsilon(x) - \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}} \geqslant \inf_{x' \in B(x, \epsilon)} f(x') - \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}}$$

$$\geqslant \lambda - \widehat{C}\epsilon^\beta - \sqrt{\frac{\log(4n) + \log(1/\delta)}{s \cdot n}} = \frac{\text{MinPts}}{v_D \cdot \epsilon^D \cdot s \cdot n}.$$

Therefore, each $x \in L_f(\lambda) \cap X_{[n]}$ is a core point. Finally, it suffices to bound the distance from any $x \in L_f(\lambda)$ to some point in $L_f(\lambda) \cap X_{[n]}$. We have for all $x \in L_f(\lambda)$ that

$$\mathcal{P}(B(x, \epsilon)) = v_D \cdot \epsilon^D \cdot f_\epsilon(x) \geqslant v_D \cdot \epsilon^D \cdot (\lambda - \widehat{C}\epsilon^\beta) \geqslant \frac{16D \cdot \log(4/\delta) \cdot \log n}{n},$$

for n sufficiently large. Then we have by Lemma 4 that with probability at least $1 - \delta/2$, $\sup_{x \in L_f(\lambda)} d(x, \widehat{L_f(\lambda)}) \leqslant \epsilon$, as desired. $\qquad \square$

B Additional Experiments

	n	D	c	s	MinPts	Range of ϵ
Australian	1,000,000	14	2	0.001	10	[0.2, 2.5)
Still	949,983	3	6	0.001	10	[0.1, 13.5)
Watch	3,205,431	3	7	0.01	10	[0.001, 0.05)
Satimage	1,000,000	36	6	0.02	10	[1, 6)
Phone	1,000,000	3	7	0.001	10	[0.001, 0.06)
Wearable [49]	165,633	16	5	0.01	10	[1, 80)
Iris	150	4	3	0.3	10	[0.1, 2.2)
LIBRAS	360	90	15	0.3	10	[0.5, 1.6)
Page Blocks	5,473	10	5	0.1	10	[1, 10,000)
kc2 [42]	522	21	2	0.3	10	[50, 7,000)
Faces	400	4,096	40	0.3	10	[6, 10)
Ozone	330	9	35	0.3	10	[100, 800)
Bank	8,192	32	10	0.01	10	[3.7, 7.4)
Ionosphere	351	33	2	0.3	10	[1, 5.5)
Mozilla	15,545	5	2	0.03	2	[1, 7,000)
Tokyo	959	44	2	0.1	2	[10K, 1,802K)
Vehicle [43]	846	18	4	0.3	10	[10, 40)

Figure 8: *Summary of all datasets used.* We give the datasets used in the main text along with eight additional real datasets. For each dataset, we give n (dataset size), D (number of features), c (number of clusters), and s (sampling rate). We tuned each dataset over 10 equally-spaced ϵ values in the range given. We chose s for each dataset from a small set of values roughly depending on the size of the dataset. In theory, s can be chosen as $O(\log n/n)$, but there is a constant multiplier that is data-dependent, and it was much simpler to tune s over a small grid, which is described in the captions of the experiment results.
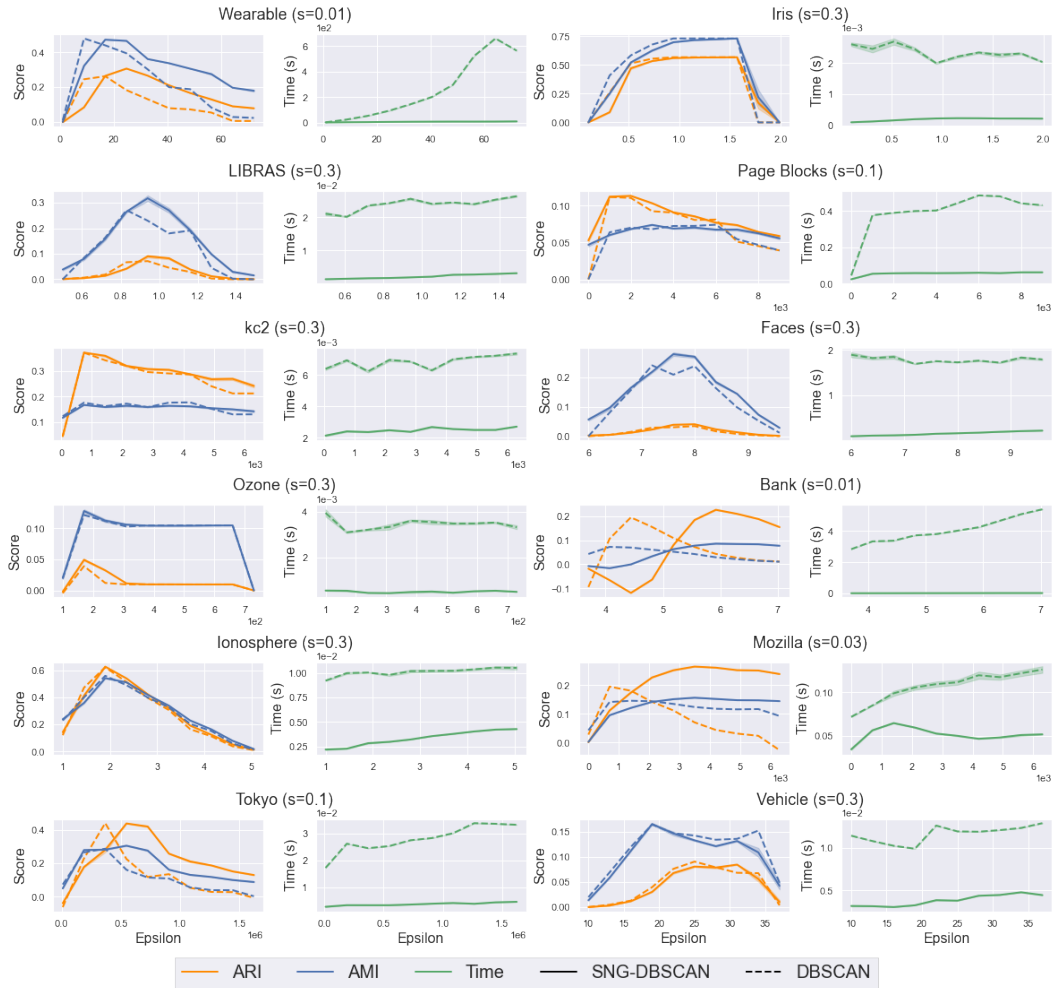
Figure 9: **Small datasets**. We show the performance of SNG-DBSCAN and DBSCAN on 12 smaller datasets. Runtimes and scores for both SNG-DBSCAN and DBSCAN are averaged over 10 runs, and 95% confidence intervals (standard errors) are shown. We ran these experiments on a local machine with 16 GB RAM and a 2.8 GHz Quad-core Intel Core i7 processor. We used $s \in \{0.01, 0.03, 0.1, 0.3\}$ depending on the size of the dataset, across a wide range of epsilon, and with MinPts $= 10$ for most datasets, other than Mozilla and Tokyo where we used MinPts $= 2$.
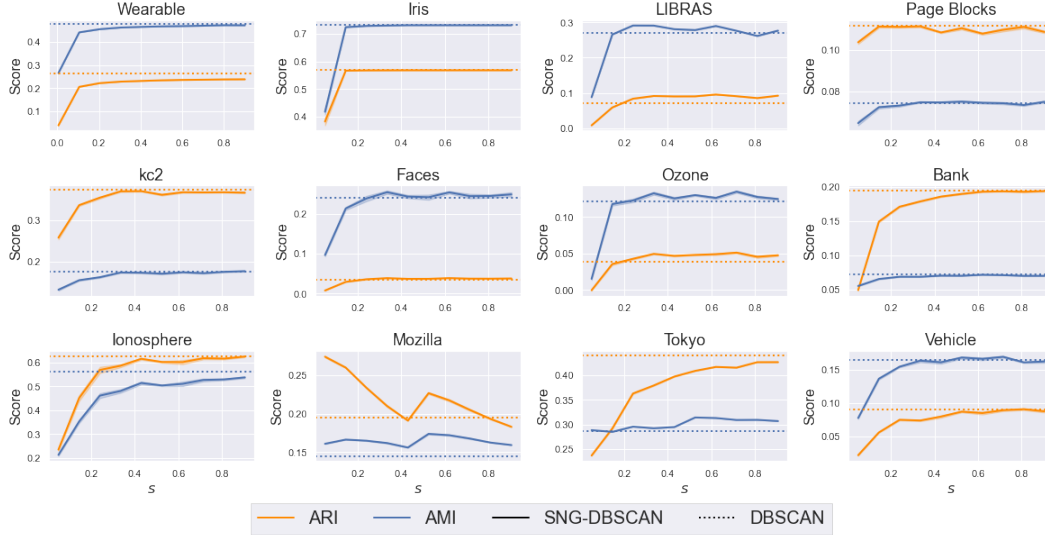
Figure 10: **Sampling rates**. We plot the performance of SNG-DBSCAN across different sampling rates s to be better understand the effect of s on the clustering quality. We make comparisons to the best performance of DBSCAN shown as the dotted vertical lines for each score. SNG-DBSCAN is run on the same epsilon as the DBSCAN benchmark for that dataset. SNG-DBSCAN values are averaged over 10 runs, and 95% confidence intervals (standard errors) are shown. We see that SNG-DBSCAN converges quickly to comparative or better clusters even at low sampling rates suggesting that often-times a small s suffices, highlighting both the stability and scalabiltiy of SNG-DBSCAN.

	Base ARI	SNG ARI	Base AMI	SNG AMI
Wearable	0.2626 (55s)	**0.3064** (8.2s)	**0.4788** (26s)	0.4720 (6.6s)
Iris	**0.5681** (<0.01s)	**0.5681** (<0.01s)	**0.7316** (<0.01s)	**0.7316** (<0.01s)
LIBRAS	0.0713 (0.03s)	**0.0903** (<0.01s)	0.2711 (0.02s)	**0.3178** (<0.01s)
Page Blocks	0.1118 (0.38s)	**0.1134** (0.06s)	**0.0742** (0.49s)	0.0739 (0.06s)
kc2	0.3729 (<0.01s)	**0.3733** (<0.01s)	**0.1772** (<0.01s)	0.1671 (<0.01s)
Faces	0.0345 (1.7s)	**0.0409** (0.16s)	0.2399 (1.7s)	**0.2781** (0.15s)
Ozone	0.0391 (<0.01s)	**0.0494** (<0.01s)	0.1214 (<0.01s)	**0.1278** (<0.01s)
Bank	0.1948 (3.4s)	**0.2265** (0.03s)	0.0721 (3.3s)	**0.0858** (0.03s)
Ionosphere	0.6243 (0.01s)	**0.6289** (<0.01s)	**0.5606** (0.01s)	0.5437 (<0.01s)
Mozilla	0.1943 (0.08s)	**0.2642** (0.05s)	0.1452 (0.09s)	**0.1558** (0.05s)
Tokyo	**0.4398** (0.02s)	0.4379 (<0.01s)	0.2872 (0.02s)	**0.3053** (<0.01s)
Vehicle	**0.0905** (0.01s)	0.0845 (<0.01s)	0.1643 (<0.01s)	**0.1653** (<0.01s)

Figure 11: *Clustering performance for additional datasets.* Best Adjusted Rand Index and Adjusted Mutual Information scores for both DBSCAN and SNG-DBSCAN under best hyperparameter tuning are given with associated runtimes in parentheses.