International Journal of Service and Knowledge Management
International Institute of Applied Informatics
2022, Vol. 6, No. 1, IJSKM644

Anytime Cell-based DBSCAN Algorithm that Connects Randomly Selected Cells and Its Performance Evaluation

Tatsuhiro Sakai [*], Keiichi Tamura [†],
Hajime Kitakami [†], Toshiyuki Takezawa [†]

Abstract

Data clustering is an unsupervised learning method that finds groups of similar features in a dataset without defining class labels. With the growing interest in big data, there is a need for techniques to speed up clustering algorithms. One clustering method that is well-known in the database domain is density-based spatial clustering of applications with noise (DBSCAN). Since the DBSCAN algorithm was first proposed, several speed-up methods have been introduced, such as the cell-based DBSCAN algorithm. The cell-based DBSCAN algorithm divides the whole dataset into smaller cells and connects them to form clusters. In this paper we propose a novel clustering algorithm called the anytime cell-based DBSCAN algorithm. The proposed algorithm connects some randomly selected cells and calculates the clustering result at high speed. The process is then repeated to improve the clustering accuracy and obtain accurate results. In this paper, we report experimental results on synthetic and real datasets showing that the proposed algorithm can calculate clustering results with high accuracy at high speed.

Keywords: DBSCAN, clustering, cell-based DBSCAN algorithm, anytime algorithm

1 Introduction

The widespread popularity of big data has resulted in the need for speed-up techniques for data clustering. Data clustering is an unsupervised learning method that can determine groups with similar features as clusters from given datasets. It is an important technique that is universally used in big data analysis because it can discover characteristic groups from datasets whose characteristics are unknown.

Density-based clustering algorithms are some of the simplest but most robust clustering techniques, automatically extracting an arbitrary number of clusters. Density-based spatial clustering of applications with noise (DBSCAN) has been proposed as a typical density-based clustering algorithm [1, 2]. The key concept of the DBSCAN algorithm is that for each data point in a cluster, the neighborhood with a user-defined radius must contain at least a given number of points, i.e., the neighborhood density must exceed a predefined

[*] Institute of Science and Engineering, Academic Assembly, Shimane University, Shimane, Japan
[†] Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan

threshold $MinPts$. If the number of data points in the neighborhood of a data point exceeds the threshold $MinPts$, that data point is called a core data point. The DBSCAN algorithm then forms clusters by connecting these core data points. The DBSCAN algorithm is used in various applications to automatically extract clusters of any shape without setting the number of clusters in advance.

The DBSCAN algorithm has two computationally expensive processes: (1) range queries to determine the neighborhood of each data point and (2) label propagation to form clusters by connecting the core data points. When the DBSCAN algorithm uses a non-indexed dataset, the computation time required for an exact algorithm is $O(n^2)$. Therefore, a number of DBSCAN algorithm speed-ups have been developed. Such works can be categorized as exact algorithms, approximate algorithms, or anytime algorithms. Recently, cell-based DBSCAN algorithms [3, 4, 5] have been developed as exact and approximate algorithms, and AnyDBC [6, 7] was developed as an anytime algorithm. These algorithms are sped up versions of the DBSCAN algorithm.

In a cell-based DBSCAN algorithm, the entire dataset is divided into cells. For two-dimensional data, each cell is a $\varepsilon/\sqrt{2} \times \varepsilon/\sqrt{2}$ square. If the number of data points in a cell is larger than $MinPts$, all data points in the cell are automatically determined to be core data points. This cell division significantly reduces the number of range queries required to determine the core data points. A cell-based DBSCAN algorithm connects cells to form clusters. Two cells are connected if and only if there is a pair of core data points within distance of ε in the cell. In a cell-based DBSCAN algorithm, clusters are formed based on the cells, which results in faster label propagation. In our previous work, we used a minimum bounding rectangle (MBR) to speed up the process of connecting cells [8, 9]. However, in this case, the clustering results cannot be output until all the processing is completed.

AnyDBC is a kind of anytime DBSCAN algorithm. An anytime algorithm outputs approximate results quickly and improves them continuously [10]. An anytime algorithm for data clustering can output clustering results of a certain accuracy at any time, finally outputting exact clustering results. AnyDBC performs a range query on randomly selected data points and calculates the clustering result at that time. It then executes range queries on some data points again to form clusters, and the accuracy of the clustering results increases as the process progresses. However, the processing time of AnyDBC is unstable, as it can be large depending on the distribution of data and randomly selected data points.

In this study, we developed an anytime cell-based DBSCAN algorithm under Euclidean distance. The anytime cell-based DBSCAN algorithm is fast, stable, and can output clustering results with consistent accuracy at any time. The contributions of this paper are as follows;

- We propose a novel clustering algorithm called the anytime cell-based DBSCAN algorithm. The proposed algorithm connects several randomly selected cells and calculates the clustering result at high speed. The proposed algorithm repeats the process of connecting randomly selected cells and calculating the clustering result to improve the accuracy of the clustering result and finally obtain the exact clustering result.

- Through evaluations on synthetic and real datasets, we demonstrate that the proposed algorithm outperforms conventional algorithms.

The remainder of this paper is organized as follows. In Section 2, related work is

reviewed. In Sections 3 and 4, we explain DBSCAN and cell-based DBSCAN, respectively. In Section 5, we present the proposed anytime cell-based DBSCAN algorithm. In Section 6, we report our experiments. Finally, Section 7 concludes the paper.

2 Related Work

Clustering techniques play an important role in the analysis of big data, and there has been much research on speeding up the process. The DBSCAN algorithm was first introduced by Ester et al. [1, 2] and applies the concept of density-based clusters. The DBSCAN algorithm has two computationally expensive processes: (1) range queries to determine the neighborhood of each data point and (2) label propagation to connect core data points to form clusters. Therefore, various DBSCAN speed-up algorithms have been developed. Works related to speeding up the DBSCAN algorithm include exact algorithms [11, 12], which guarantee exact clustering results, approximate algorithms [13, 14, 15, 16, 17, 18, 19], which yield approximate clustering results, and anytime algorithms, which quickly output approximate clustering results and finally output exact clustering results. In general, approximate algorithms are faster than exact algorithms, but the clustering results obtained may differ from the exact clustering results, depending on the parameters and data distribution.

Recently, the cell-based DBSCAN algorithm [3, 4, 5, 8, 9] has been developed as an exact and approximate algorithm. The cell-based DBSCAN algorithm can significantly reduce the number of range queries and speed up label propagation because it forms clusters based on cells. However, the clustering results cannot be output until all the processing is complete. Therefore, the usability must be improved by outputting provisional clustering results even if the processing is not completed.

AnyDBC was proposed as an anytime DBSCAN algorithm [6, 7]. AnyDBC performs range queries on a few randomly selected data points and calculates the clustering results at that time. It then repeats this process efficiently to form clusters. The clustering results of AnyDBC become increasingly accurate as the process is repeated. Moreover, AnyDBC shows that by not performing a range query on a data point that does not change the clustering result, the number of range queries can be reduced and the exact clustering result can be approached efficiently.

However, the processing time of AnyDBC is unstable and can be large depending on the distribution of the dataset and randomly selected data points. In addition, AnyDBC requires a large number of range queries, so it is necessary to construct an index structure such as a *kd*-tree. In other words, AnyDBC requires more memory than the cell-based DBSCAN algorithm. In this paper, we propose a novel anytime cell-based DBSCAN algorithm with fast and stable processing time. To the best of our knowledge, this is the first time cell-based DBSCAN has been extended to an anytime algorithm.

3 DBSCAN

In this section, the definitions used for DBSCAN are briefly reviewed. Let DP be a set of data points in a d-dimensional space. In DBSCAN, the ε-neighborhood of a data point is defined as the data points in the neighborhood of a user-defined radius ε.

Definition 1 (ε-neighborhood $N_\varepsilon(dp)$)**.** The ε-neighborhood of data point dp, denoted by

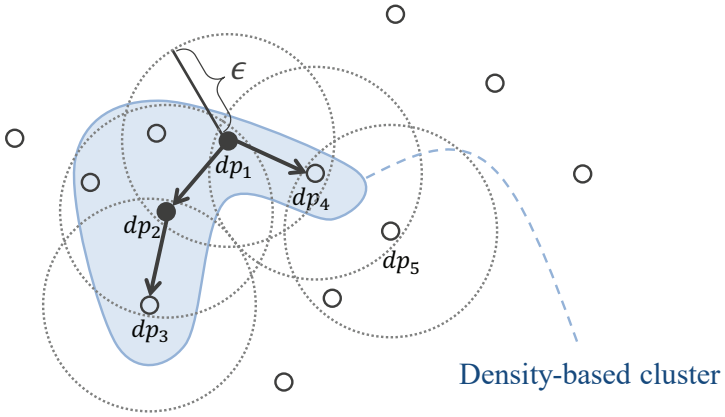

Figure 1: Example of definitions ($|N_\varepsilon(dp_1)| = 4$, $|N_\varepsilon(dp_2)| = 5$, $|N_\varepsilon(dp_4)| = 3$)

$N_\varepsilon(dp)$, is defined as

$$N_\varepsilon(dp) = \{dq \in DP | dist(dp,dq) \leq \varepsilon\}, \tag{1}$$

where the function *dist* returns the Euclidean distance between data points dp and dq.

Definition 2 (Core data point and Border data point)**.** A data point dp is called a core data point if there is at least a given number of data points, $MinPts$, in the ε-neighborhood $N_\varepsilon(dp)$ ($|N_\varepsilon(dp)| \geq MinPts$). Otherwise, if $|N_\varepsilon(dp)| < MinPts$, dp is called a border data point.

Definition 3 (ε -Density-based reachable)**.** Suppose that there is a data point sequence $(dp_1, dp_2, \cdots, dp_n)$. Data point dp_n is ε -density-based reachable from dp_1 if there is a sequence such that

(1) $dp_1, dp_2, \cdots, dp_{n-1}$ are core data points and

(2) dp_{i+1} is in the ε-neighborhood of dp_i ($dp_{i+1} \in N_\varepsilon(dp_i)$).

An illustrative example of the above definitions is shown on Figure 1. Suppose that $MinPts = 4$. In Figure 1, data points dp_1 and dp_2 are core data points because $N_\varepsilon(dp_1) = 4$ and $N_\varepsilon(dp_2) = 5$. Therefore, dt_2 is ε-densely-based reachable from dt_1, and dt_3, which is in the ε-neighborhood of dt_2, is also ε-densely-based reachable from dt_1. In contrast, the data point dp_4 is a border data point because $N_\varepsilon(dp_4) = 3$. It is ε-density-based reachable from dt_1 to dt_4, but not ε-density-based reachable from dt_1 to dt_5.

In DBSCAN, density-based clusters are formed by recursively connecting data points that are ε-density-based reachable from the core data point. Data points that are not ε-density-based reachable from any core data point are called noise. A density-based cluster

consists of two types of data: core data points, which are mutually density-based reachable, and border data points, which are in the ε-neighborhood of the core data point. A density-based cluster is defined as follows.

Definition 4 (Density-based cluster)**.** A density-based cluster *DC* in a data point set satisfies the following restrictions:

(1) $\forall dp,\ dq \in DP$, dq is in *DC* if and only if $dp \in DC$ and dq is ε-density-based reachable from dp.

(2) $\forall dp,\ dq,\ do \in DC$, dp and dq are ε-density-based reachable from do.

In the example of Figure 1, the data points surrounded in blue form a density-based cluster.

4 Cell-based DBSCAN

In this section, we present cell-based DBSCAN.

4.1 Overview

Cell-based DBSCAN is divided into four main steps: cell division, determining the core data points, connecting cells, and determining the border data points or noise. When all steps are complete, all data points are classified into a cluster or as noise. An outline of these four main steps is as follows:

- In the cell division step, the whole dataset is divided into cells. Cell-based DBSCAN performs clustering based on the divided cells.
- In the core data point determination step, each data point is analyzed to verify if it is a core or non-core data point.
- In the connecting cells step, cell-based DBSCAN connects the neighborhood cells to form clusters.
- In the border data point or noise determination step, all non-core data points are analyzed to verify if they are border data points or noise.

4.2 Cell Division

In the first step, cell-based DBSCAN divides the whole dataset into small cells, where each cell is a $\varepsilon/\sqrt{d} \times \varepsilon/\sqrt{d}$ square. This cell size is chosen because it facilitates the determination of whether a cell is dense. Then, the algorithm assigns each data point to a cell. Figure 2 shows an example of cell-based DBSCAN. In Figure 2, the dataset is divided into $\varepsilon/\sqrt{2} \times \varepsilon/\sqrt{2}$ because $d = 2$.

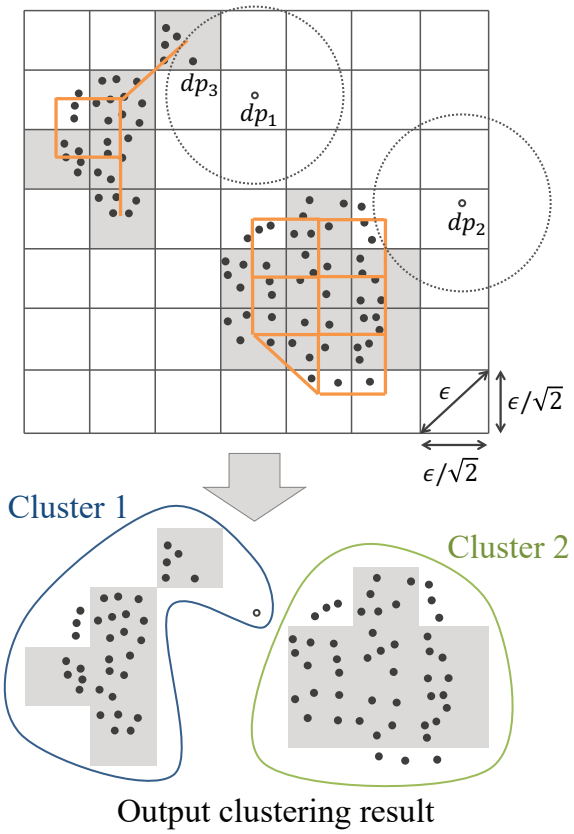

Figure 2: Example of cell-based DBSCAN

4.3 Core Data Point Determination

This step determines whether each data point is a core or non-core data point. If the number of data points in a cell C_i is $|C_i| \geq MinPts$, all data points in C_i are determined to be core data points automatically because the distance between data points in the same cell is at most ε. In contrast, if $|C_i| < MinPts$, data points belonging to cell C_i will be determined as core or non-core by calculating the distance to the data points in the neighborhood cells. In Figure 2, where $MinPts = 4$, there are 12 cells that exceed the value of $MinPts$, as indicated by gray shading. All data points in these cells are determined to be core data points. A data point in another cell is determined to be a core or non-core point by calculating the distance between itself and the data points in the neighborhood cells.

4.4 Connecting Cells

In this step, neighborhood cells are connected to form clusters. These cells are connected if and only if an arbitrary pair of core data points in two cells are within ε. Cell-based DBSCAN checks whether cells are connected for all cells containing core data points. The data points in two connected cells are ε-densely-based reachable, and the data point set is a density-based cluster. In Figure 2, two clusters are formed on the upper left side and

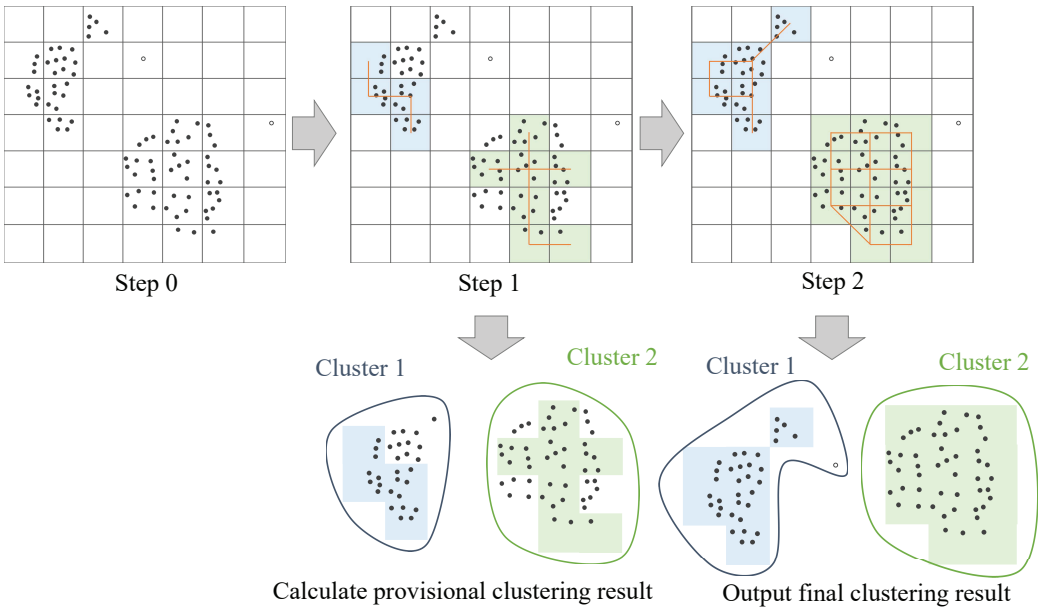

Figure 3: Example of connecting randomly selected cells

lower right side. Our proposed algorithm performs high-speed checking using an MBR, as proposed in [8, 9].

4.5 Border Data Point or Noise Determination

Finally, cell-based DBSCAN determines if the non-core data points are border data points or noise. If there is at least one core data point in the ε-neighborhood of a data point $N_\varepsilon(dp_i)$, it belongs to the cluster of the core data point because dp_i is a border data point. In contrast, a data point dp_i is noise if there are no core data points in $N_\varepsilon(dp_i)$. In Figure 2, a core data point dp_3 can be observed in $N_\varepsilon(dp_1)$. That is, dp_1 belongs to the cluster of the core data point dp_3. In contrast, there are no core data points in $N_\varepsilon(dp_2)$. Therefore, dp_2 does not belong to a cluster and is deemed to be noise. After processing this step, cell-based DBSCAN outputs the clustering result, as shown in Figure 2.

5 Proposed Algorithm

In this section, we present the proposed anytime cell-based DBSCAN algorithm.

5.1 Overview

The proposed anytime cell-based DBSCAN algorithm connects some randomly selected cells. Then, it determines whether the data points in the cells other than the connected cells are border data points or noise and calculates a clustering result. This process is repeated multiple times; with each repetition, the clustering accuracy improves, and it finally yields

Algorithm 1 Anytime cell-based DBSCAN algorithm

Input: DP, d, ε, $MinPts$, p
Output: $ClusterList$, $NoiseList$

```
1: /*Cell division*/
2: Create CellList by dividing a set of data points into cells, where each cell is an ε/√d × ε/√d square
3: /*Core data point determination*/
4: for i = 0 to |CellList| do
5:    Determine whether each data point in a cell C_i is a core or non-core data point
6: end for
7: Create a cell pair list CellPairList
8: for i = 0 to p do
9:    ClusterList = φ, NoiseList = φ
10:   /*Connecting randomly selected cells*/
11:   for j = 0 to |CellPairList|/p do
12:      Randomly select a cell pair CP from CellPairList
13:      If it is determined that the cell pair CP can be connected, then it is connected
14:   end for
15:   Create ClusterList, i.e., a cluster set based on data points in the connected cells
16:   /*Border data point or noise determination*/
17:   for j = 0 to |DP| do
18:      if dp_j does not belong to a cluster and N_ε(dp_j) has a core data point belonging to a cluster then
19:         Add dp_j to the cluster of the core data point
20:      else
21:         Add dp_j to NoiseList
22:      end if
23:   end for
24:   Save ClusterList and NoiseList as the current clustering result
25: end for
26: return ClusterList, NoiseList
```

the exact clustering result. If the user stops the algorithm to get the clustering result during the repetitions, the clustering result calculated at that instance is given as the output.

5.2 Connecting Randomly Selected Cells

In this section, we describe our method to connect the randomly selected cells used by the proposed algorithm. Let p be a parameter denoting the number of times that the clustering result is calculated. First, the proposed algorithm creates a cell pair list $CellPairList$. Then, a cell pair is randomly selected from $CellPairList$, and the proposed algorithm determines whether the cell pair can be connected. If the cell pair can be connected, it is. Each cell pair is selected $|CellPairList|/p$ times, and the proposed algorithm determines whether it is a data point or noise; finally, it calculates the clustering result. The clustering result calculated at the p-th instance has already checked the connection between all cells of $CellPairList$; therefore, it outputs the exact clustering result.

Figure 3 shows an example of connecting randomly selected cells. In this example,

Table 1: Detail of datasets

Dataset	Dimensions	Number of Data Points
SSD2	2	10,000,000
SSD3	3	10,000,000
SSD5	5	10,000,000
SSD7	7	10,000,000
PAMAP2	4	3,850,505
FARM	5	3,627,086
HOUSEHOLD	7	2,049,280

$p = 2$. Some randomly selected cells are connected to form two clusters in Step 1. The proposed algorithm determines the border data points or noise from the remaining data points in Step 1, and outputs the provisional clustering result. During Step 2, if the user stops the algorithm, the provisional clustering result calculated in Step 1 is output. In Step 2, all cell pairs have been checked and connected, and the final exact clustering result is obtained.

5.3 Algorithm

In this section, we describe the overall flow of the proposed algorithm, as shown in Algorithm 1. Algorithm 1 receives a set of data points DP in a d-dimensional space, along with the parameters ε, $MinPts$, and p. Then, it outputs a cluster list $ClusterList$ and noise list $NoiseList$. First, the proposed algorithm divides a set of data points into cells, where each cell is a $\varepsilon/\sqrt{d} \times \varepsilon/\sqrt{d}$ square (line 2). Next, the core data points are determined (lines 4 to 6). Then, the algorithm creates $CellPairList$ (line 7). After that, it connects the randomly selected cells, and creates $ClusterList$, which is a cluster set based on data points in the connected cells (lines 8 to 15). Finally, it determines the border data points and noise and creates $NoiseList$ (lines 17 to 23). The proposed algorithm repeats this process p times (lines 8 to 25). If the user stops the algorithm to get the clustering result during the processing, $ClusterList$ and $NoiseList$ saved at that instance (line 24) are given as outputs. If the algorithm is executed the completion, the final exact $ClusterList$ and $NoiseList$ are given as outputs (line 26).

6 Experiments

In this section, we report our experiments.

6.1 Setups

To evaluate the proposed algorithm, we conducted experiments using synthetic and real datasets. We compared the results obtained using the proposed anytime cell-based DBSCAN algorithm (denoted by AnyCDBC), exact cell-based DBSCAN algorithm with the MBR criteria (denoted by CDBC) [9], and the anytime algorithm with single processing for DBSCAN (denoted by AnyDBC) [6]. We analyzed the processing times and normalized mutual information (NMI) [20] to compare them with the ground truths. We implemented

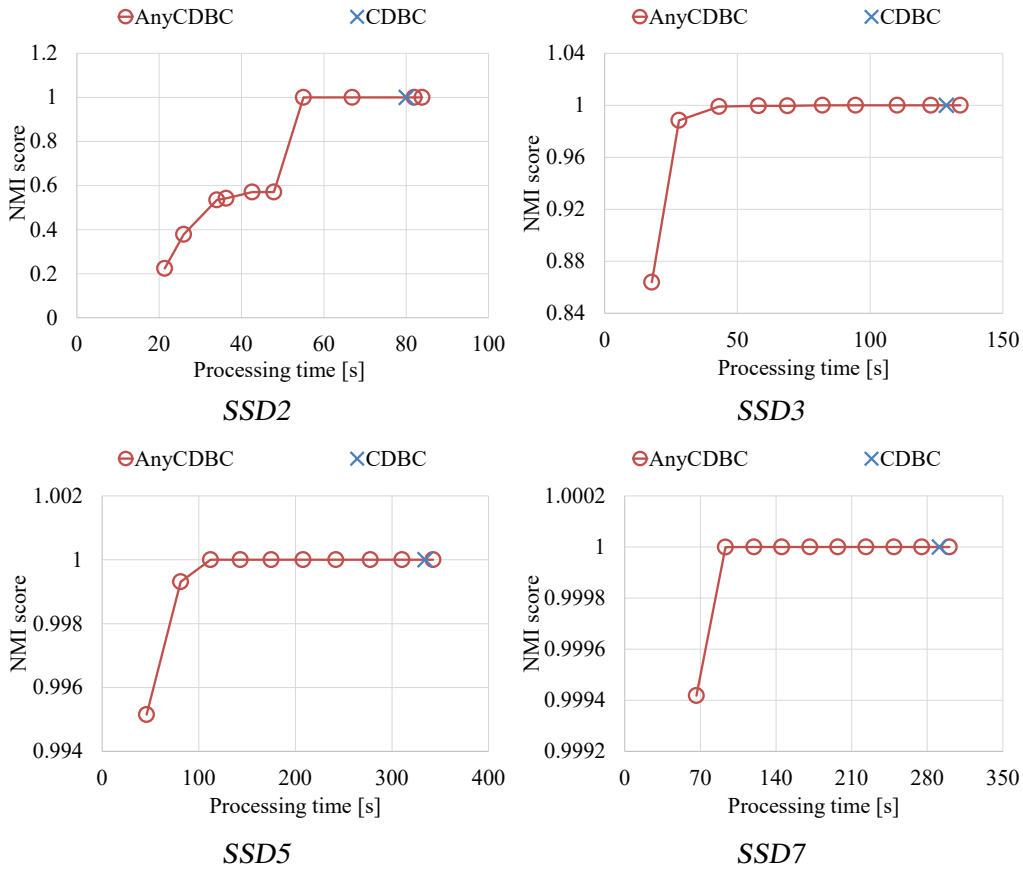

Figure 4: Experimental results of AnyCDBC and CDBC using synthetic datasets

AnyCDBC and CDBC using C++, g++ 7.4.0 as the compiler, and -O3 as the option on a PC with an Intel Core i7-8700 @3.2 GHz CPU and 16 GB RAM. We used the binary file of AnyDBC provided by the authors.

In the experiments, we used both synthetic datasets generated by the data generator scheme of [4] and real datasets. Table 1 lists the details of each dataset. We used synthetic datasets *SSD2*, *SSD3*, *SSD5*, and *SSD7* of dimensions d = 2, 3, 5, and 7, respectively, each with 10,000,000 data points. Real datasets *PAMAP2*[21], *FARM*[22], and *HOUSEHOLD*[23] of dimensions d = 4, 5, and 7, respectively, were used. The distribution of synthetic datasets has some dense parts, and the amount of noise is small. The distribution of the real datasets is relatively sparse, and the amount of noise is large. The parameters of AnyCDBC, CDBC, and AnyDBC were set to $MinPts = 100$, and ε was varied from 5,000 until the clustering result was a single cluster. The parameter p of AnyCDBC was set to $p = 10$.

6.2 Experimental Results using Synthetic Datasets

Figure 4 shows the experimental results of AnyCDBC and CDBC using the synthetic datasets with $\varepsilon = 5,000$. The experimental results of AnyCDBC show transitions in the NMI score of the clustering result calculated ten times because $p = 10$. From the exper-

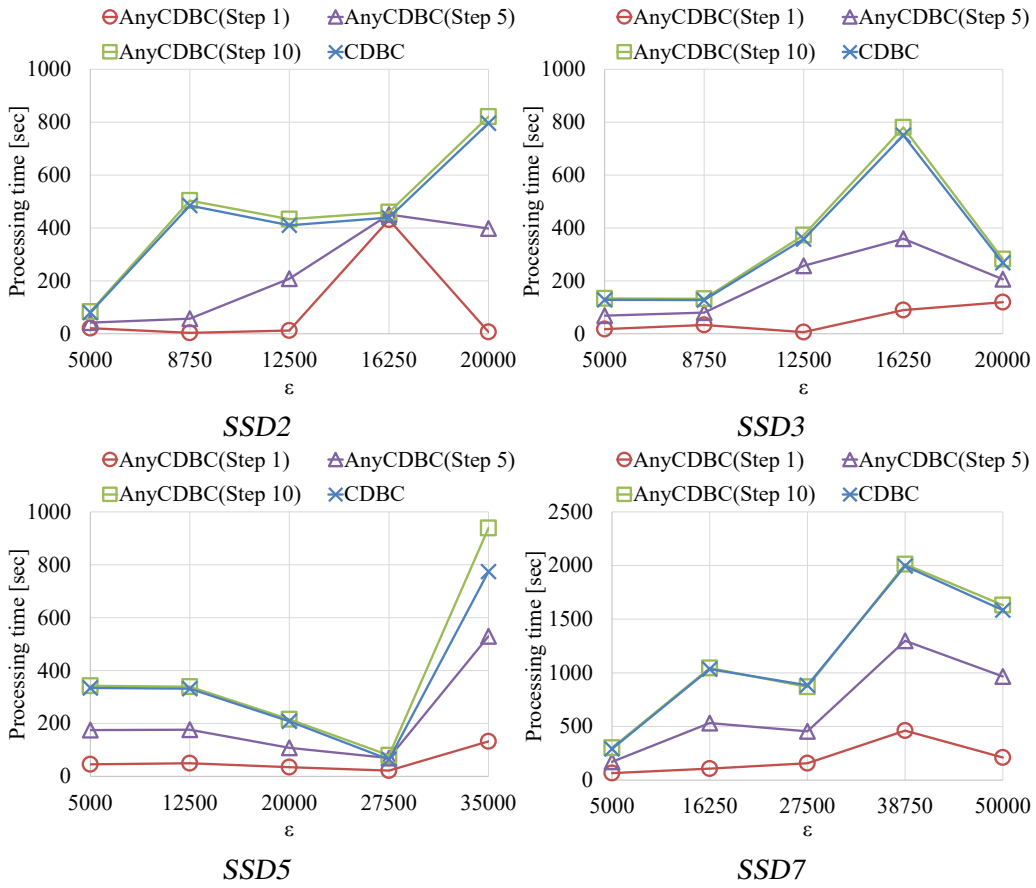

Figure 5: Experimental results of AnyCDBC and CDBC using synthetic datasets under the influence of ε

imental results of CDBC, the NMI score was 1.00 because CDBC is the exact DBSCAN algorithm. In Figure 4, it can be observed that AnyCDBC could calculate the eighth clustering result of *SSD2*, while for the other datasets, it could calculate the ninth clustering result faster than CDBC. The eighth clustering result of *SSD2* and ninth clustering result of other datasets are already the exact clustering results. That is, these results demonstrate that AnyCDBC could calculate the exact clustering results faster than CDBC. Moreover, the NMI score of the first clustering result for *SSD3* was low; the first to sixth results for *SSD2* were also low. However, the other clustering results were the same as the exact clustering results, or the NMI score was over 0.98. Owing to the use of synthetic datasets, it was found that AnyCDBC can calculate almost exact clustering results at high speed.

Figure 5 shows the experimental results of AnyCDBC and CDBC using synthetic datasets under the influence of ε. The experimental results of AnyCDBC show the processing time to calculate the clustering results for the first, fifth, and final tenth times. In Figure 5, the processing time for calculating the first and fifth results of AnyCDBC is faster than the processing time of CDBC. In addition, there is no significant difference in processing time when comparing the final tenth results of AnyCDBC and CDBC. Even when ε is varied, AnyCDBC is able to calculate clustering results close to the exact clustering result at high

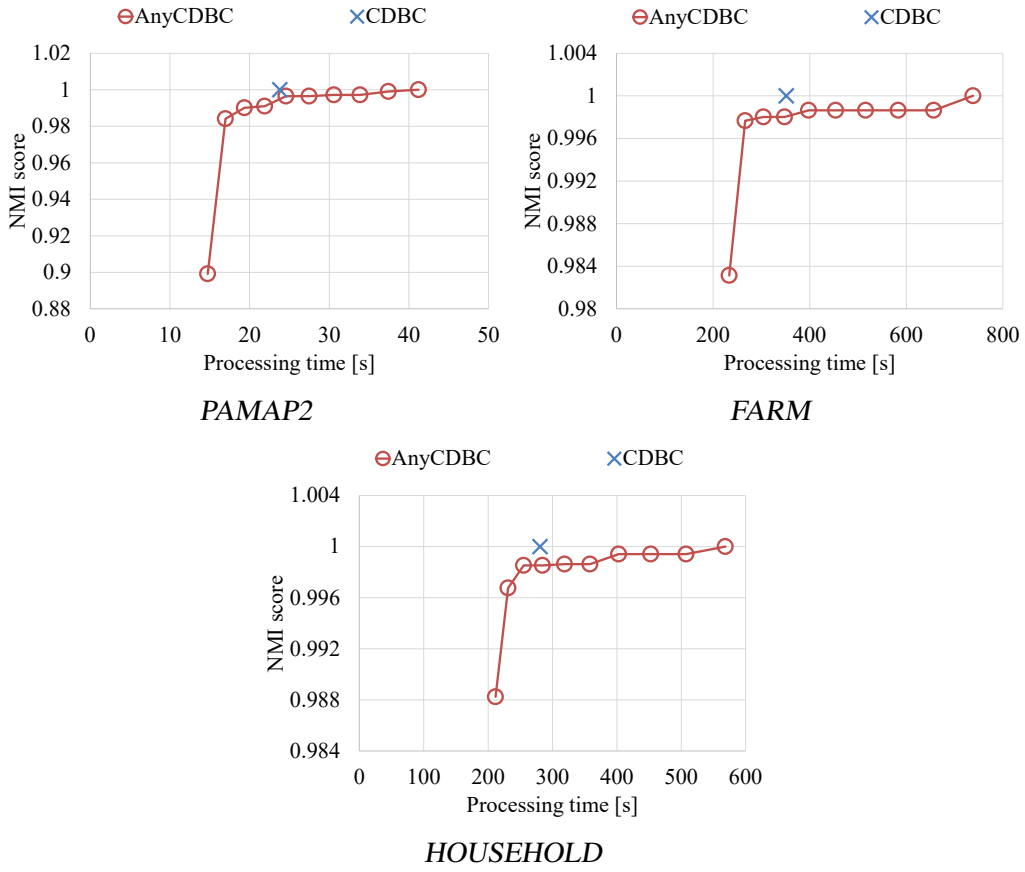

Figure 6: Experimental results of AnyCDBC and CDBC using real datasets

speed, while the final clustering result is almost equal to CDBC in processing time.

In the experimental results on *SSD2* with $\varepsilon = 16250$ in Figure 5, the processing time of the first clustering result of AnyCDBC is almost equal to that of the tenth clustering result. The reason for this is that there was a pair of cells that took 407s to determine if the cells were connected or not. This connecting decision occupied most of the total processing time, and the connecting decision was performed before the first clustering result was calculated. In this manner, when a large amount of processing time is required for a pair of cells, there is no difference in processing time between the provisional and final clustering results.

6.3 Experimental Results using Real Datasets

Figure 6 shows the experimental results with $\varepsilon = 5000$ for AnyCDBC and CDBC using real datasets. In Figure 4, it can be observed that AnyCDBC can calculate the fourth clustering results for *PAMAP2* and *FARM* and the third clustering result for *HOUSEHOLD* faster than CDBC. The NMI scores of the fourth clustering results for *PAMAP2* and *FARM* and the third clustering results for *HOUSEHOLD* are over 0.99. In this case, totals of 271, 12, and 16 data points were misclassified as noise on PAMAP2, FARM, and HOUSEHOLD, respectively. AnyCDBC showed that almost exact clustering results can be calculated at high speed.

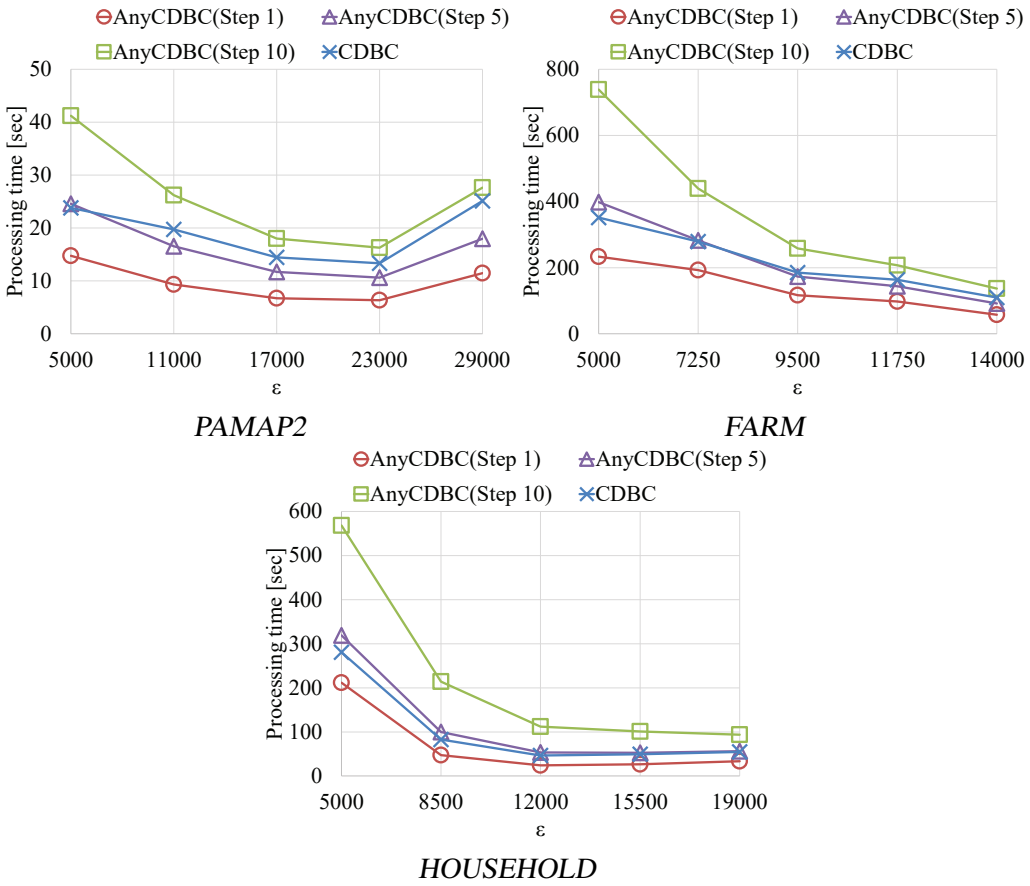

Figure 7: Experimental results of AnyCDBC and CDBC using real datasets under the influence of ε

However, on the real datasets, the processing time for the final clustering results using AnyCDBC was much slower than that using CDBC. Significant processing time was required to determine border data points and noise, which is performed every time the clustering result is calculated because the real datasets are sparsely distributed and contain a large amount of noise. If the number of times p to calculate the clustering result is small, this problem can be solved, but the usefulness of anytime clustering algorithm will decrease. In the proposed algorithm, the clustering result is calculated by connecting some cells.

Figure 7 shows the experimental results of AnyCDBC and CDBC using real datasets under the influence of ε. In Figure 7, the processing time for calculating the first clustering results of AnyCDBC is faster than that of CDBC, and there is no significant difference between the fifth clustering results of AnyCDBC and CDBC. Moreover, the larger ε is, the smaller the difference between the final clustering results of AnyCDBC and CDBC. This is because the larger ε is, the less noise there is, and the processing time for the border data points and noise process is reduced. AnyCDBC proved that more effective for larger ε when the dataset is sparsely spread over the whole space, such as in real datasets.

Table 2: Experimental results of AnyDBC

	SSD2	*SSD3*	*SSD5*	*SSD7*	*PAMAP2*	*FARM*	*HOUSEHOLD*
1	5900.6	44406	86.25	634.71	67.48	207.26	41.07
2	5995.1	44407	86.76	636.92	67.12	206.33	40.93
3	92694	50878	29.31	300.05	66.75	13855	40.18
4	201.06	47579	27.49	224.09	65.78	238.69	40.34
5	29248	51683	584.92	350.95	67.09	126.01	42.19
6	2066.0	81659	67.03	321.90	71.01	115.11	39.95
7	902.57	41263	33.09	348.08	73.52	112.51	40.44
8	861.77	61320	84.07	461.15	72.85	145.40	44.34
9	795.89	40832	35.72	351.06	65.41	107.74	41.33
10	792.21	42476	113.95	104.06	70.20	129.32	40.57

6.4 Experimental Results of AnyDBC

Table 2 lists the processing times using $\varepsilon = 5000$ required to calculate the final clustering results for AnyDBC. In Table 2, the processing time for AnyDBC is for ten runs because the processing times varied significantly between runs. AnyDBC is faster than AnyCDBC on the *HOUSEHOLD* dataset, but its processing time for other datasets is either faster or slower. For example, the processing times on *SSD2*, *SSD3*, and *FARM* are not stable for each run because if a large amount of data cannot be covered by a range query performed by randomly selecting the data, the number of range queries increases and more processing time is required. In particular, the processing time required by AnyDBC on *SSD2* and *SSD3* was significantly greater than that required by AnyCDBC. AnyCDBC randomly selects cell pairs for connecting, but because the number of cell pairs to be connected is equal for each execution of the algorithm, there is no significant difference in the final processing time for each run of the algorithm. Thus, AnyCDBC can execute the anytime algorithm for DBSCAN faster and more stable than AnyDBC.

7 Conclusion

This paper proposed a novel clustering algorithm called anytime cell-based DBSCAN. The proposed algorithm connects some randomly selected cells and calculates the clustering result at high speed. It repeats this process, and with each subsequent repetition, the accuracy of the clustering result improves until it becomes exact. The previous cell-based DBSCAN algorithm [9] cannot output until all the processing is complete. The proposed algorithm improves the usability by outputting provisional clustering results even if the processing is not completed.

Experimental results showed that the proposed algorithm can quickly calculate clustering results close to the exact ones when using synthetic datasets, while the final clustering results can be calculated in almost the same processing time as the previous exact cell-based DBSCAN algorithm [9]. The proposed algorithm using real datasets was able to quickly calculate clustering results close to the exact ones, but the final clustering results took much more processing time to obtain than for the previous exact cell-based DBSCAN. However, the larger the parameter ε is, the more effective the proposed algorithm is on real datasets.

The processing time of the previous anytime DBSCAN algorithm [6] is unstable and can be large depending on the distribution of the dataset and randomly selected data points, but the proposed algorithm was faster and more stable than the previous anytime DBSCAN algorithm.

In our future work, we intend to evaluate the proposed algorithm using other real datasets and develop an algorithm to calculate the final clustering result using real datasets at high speed. If the algorithm can determine noise early in the process, the algorithm will be faster. Moreover, we intend to develop a parallel processing method for the proposed algorithm.

References

[1] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. KDD 1996*, 1996, pp. 226–231.

[2] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 169–194, 1998.

[3] A. Gunawan, "A faster algorithm for DBSCAN," Master's thesis, Technische University Eindhoven, 40 pages, 2013.

[4] J. Gan and Y. Tao, "DBSCAN revisited: Mis-claim, un-fixability, and approximation," in *Proc. SIGMOD 2015*, 2015, pp. 519–530.

[5] ——, "On the hardness and approximation of euclidean dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 14:1–14:45, 2017.

[6] S. T. Mai, I. Assent, and M. Storgaard, "AnyDBC: An efficient anytime density-based clustering algorithm for very large complex datasets," in *Proc. KDD 2016*, 2016, pp. 1025–1034.

[7] S. Mai Thai, I. Assent, J. Jacobsen, and M. Dieu, "Anytime parallel density-based clustering," *Data Mining and Knowledge Discovery*, vol. 32, pp. 1121–1176, 04 2018.

[8] T. Sakai, K. Tamura, and H. Kitakami, "Cell-based DBSCAN algorithm using minimum bounding rectangle criteria," in *Proc. BDMS 2017*, 2017, pp. 133–144.

[9] T. Sakai, K. Tamura, H. Kitakami, and T. Takezawa, "Speed-up of cell based DBSCAN using minimum bounding rectangle and recursive cell partitioning," *The IEICE transactions on information and systems*, vol. J101-D, no. 4, 2018.

[10] S. Zilberstein, "Using anytime algorithms in intelligent systems," vol. 17, pp. 73–83, 1996.

[11] Y. El-Sonbaty, M. A. Ismail, and M. Farouk, "An efficient density based clustering algorithm for large databases," in *Proc. ICTAI 2004*, 2004, pp. 673–677.

[12] S. Mahran and K. Mahar, "Using grid for accelerating density-based clustering," in *Proc. CIT 2008*, 2008, pp. 35–40.

[13] S. Zhou, A. Zhou, J. Cao, J. Wen, Y. Fan, and Y. Hu, "Combining sampling technique with DBSCAN algorithm for clustering large spatial databases," in *Proc. PAKDD 2000*, 2000, pp. 169–172.

[14] M. Dash, H. Liu, and X. Xu, "'1+1>2': merging distance and density based clustering," in *Proc. DASFAA 2001*, 2001, pp. 32–39.

[15] X. Wang and H. J. Hamilton, "DBRS: A density-based spatial clustering method with random sampling," in *Proc. PAKDD 2003*, 2003, pp. 563–575.

[16] B. Borah and D. K. Bhattacharyya, "An improved sampling-based DBSCAN for large spatial databases," in *Proc. ICISIP 2004*, 2004, pp. 92–96.

[17] B. Liu, "A fast density-based clustering algorithm for large databases," in *Proc. ICMLC 2006*, 2006, pp. 996–1000.

[18] C.-F. Tsai and C.-T. Wu, "GF-DBSCAN: A new efficient and effective data clustering technique for large databases," in *Proc. MUSP 2009*, 2009, pp. 231–236.

[19] Z. Wang, R. Zhang, J. Qi, and B. Yuan, "Dbsvec: Density-based clustering using support vector expansion," in *Proc.of ICDE 2019*, 2019, pp. 280–291.

[20] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Is a correction for chance necessary?" in *Proc. ICML 2009*, 2009, pp. 1073–1080.

[21] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Proc. ISWC 2012*, 2012, pp. 108–109.

[22] M. Varma and A. Zisserman, "Texture classification: are filter banks necessary?" in *Proc. CVPR 2003*, vol. 2, 2003, pp. II–691–8 vol.2.

[23] B. K and L. M, "UCI machine learning repository," 2013.