An Algorithm for Discovering Clusters of Different Densities or Shapes in Noisy Data Sets

Fereshte Khani
Dept. of Computer Engineering
Sharif University of Technology
fkhani@ce.sharif.edu

Mohmmad Javad Hosseini
Dept. of Computer Engineering
Sharif University of Technology
mjhosseini@ce.sharif.edu

Ahmad Ali Abin
Dept. of Computer Engineering
Sharif University of Technology
abin@ce.sharif.edu

Hamid Beigy
Dept. of Computer Engineering
Sharif University of Technology
beigy@sharif.edu

ABSTRACT

In clustering spatial data, we are given a set of points in \( \mathbb{R}^n \) and the objective is to find the clusters (representing spatial objects) in the set of points. Finding clusters with different shapes, sizes, and densities in data with noise and potentially outliers is a challenging task. This problem is especially studied in machine learning community and has lots of applications. We present a novel clustering technique, which can solve mentioned issues considerably. In the proposed algorithm, we let the structure of the data set itself find the clusters, this is done by having points actively send and receive feedbacks to each other.

The idea of the proposed method is to transform the input data set into a graph by adding edges between points that belong to the same cluster, so as connected components correspond to clusters, whereas points in different clusters are almost disconnected. At the start, our algorithm creates a preliminary graph and tries to improve it iteratively. In order to build the graph (add more edges), each point sends feedback to its neighborhood points. The neighborhoods and the feedback to be sent are determined by investigating the received feedbacks. This process continues until a stable graph is created. Henceforth, the clusters are formed by post-processing the constructed graph. Our algorithm is intuitive, easy to state and analyze, and does not need to have lots of parameter tuning. Experimental results show that our proposed algorithm outperforms existing related methods in this area.

Categories and Subject Descriptors

I.5.3 [Pattern Recognition]: Clustering—Algorithm

General Terms

Algorithms, Performance

Keywords

clustering, density based, different densities, noises, outliers, spatial data

1. INTRODUCTION

Clustering is the task of partitioning data into similar groups based on similarity criteria. Clustering problems arise in a wide range of applications like information retrieval, data mining, gene analysis, computer graphics, and many others. Clusters can have variety of shapes, sizes, and densities which makes the task more challenging. For more information on clustering please refer to [11, 12]. Most clustering algorithms handle clusters with different sizes and shapes; however, there are few algorithms which consider clusters with variant densities. DBSCAN [10] (further improved in [15]) is one of the most famous algorithms for clustering. It relies on the density based notion of clusters which is designed to discover clusters with arbitrary shapes.

Basic density based clustering algorithms assume that the clusters are high density regions separated by low density regions. They can easily find clusters with different sizes and shapes. One of the most evident drawbacks of these algorithms is their weakness to recognize clusters with variant densities. As an illustration, Fig. 1 shows two clusters with different densities. However, basic density based clustering algorithms treat all of the points as one cluster.

In general, these algorithms use a global neighborhood radius, namely \( \epsilon \), to determine whether two points are in the same cluster or not. Therefore, they might merge two near clusters with different densities together. To avoid this problem if a smaller value of \( \epsilon \) is chosen, sparser clusters
will be over clustered, i.e. they are recognized as smaller separate clusters.

DDSC [6] and DD-DBSCAN [5] find the number of points in the \( \epsilon \)-neighborhood of each point. Two points then lie on the same cluster, only if their corresponding numbers are similar. Although, these algorithms are successful in many conditions, they cannot cluster appropriately when some clusters are extremely sparser than the others, like Fig. 2. In particular, as they use a global \( \epsilon \), they will either over cluster or under cluster (merge some clusters together).

In this paper, we present a novel algorithm to solve these issues. In this algorithm, instead of using \( \epsilon \), we assign a specific number to each point called level, which is used to construct an adjacency graph for clustering. However, level values are not known to the algorithm as a priori and are dependent to the graph, itself. At the beginning of the algorithm, we construct a preliminary graph without any edge and estimate the level values incrementally as the algorithm proceeds via active communication of feedback among the points. Therefore, as a predetermined global \( \epsilon \) is not used, the problems occurred for DDSC or DD-DBSCAN cannot be seen anymore. As a result, the algorithm can accurately detect clusters with high differences in their densities.

The rest of the paper is organized as follow. In the next section, we overview the related works and introduce basic concepts of density-based clustering algorithms. In section 3, the proposed algorithm is introduced. In Section 4, we evaluate the proposed algorithm and compare the experimental results to some previous works. Finally, in section 5 a conclusion and some directions for future works are given.

2. RELATED WORK

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [10] algorithm was introduced to find clusters using density based clustering approach. DBSCAN classifies points in two separate classes core points and border points according to two parameters: \( \epsilon \) and MinPts. To be more precise, \( \epsilon \)-neighborhood of a point \( p \) is a ball centered at \( p \) with radius \( \epsilon \). If the \( \epsilon \)-neighborhood of \( p \) contains at least MinPts of points, then it marks \( p \) as a core point; otherwise it marks \( p \) as a border point. After classification of points, it starts with a core point \( q_0 \) and assign all the points which are reachable from \( q_0 \) to the same cluster. Where point \( q \) is reachable from \( q_0 \) if there is a sequence \( q_0, q_1, q_2, \ldots, q_n \) of core points in which for all \( (1 \leq i \leq n) \), \( q_i \) lies in \( \epsilon \)-neighborhood of \( q_{i-1} \) and \( q \) lies on \( \epsilon \)-neighborhood of \( q_n \). The algorithm continues this procedure for all the non-visited core points.

Although this algorithm has many features and can recognize clusters with different shapes and sizes but has some drawbacks as well. There are lots of algorithms trying to improve DBSCAN, such as ST-DBSCAN [3] which considers temporal data in addition to the spatial data, IDBSCAN [4] which enhances the speed of DBSCAN, DD-DBSCAN [5] and DDSC [6] which improve DBSCAN in the sense that it finds clusters with different densities.

DDSC and DD-DBSCAN mark point \( p \) as a core point if it satisfies some other properties too. To be more precise, DD-DBSCAN mark point \( p \) with \( m_0 \) points in its \( \epsilon \)-neighborhood as a core point if the neighborhood ball of all \( m_0 \) points are similar to the \( p \). DDSC uses similar ideas. In spite of their ability to find clusters with different densities better than DBSCAN, they still perform poorly in finding clusters with enormous difference in their densities.

There are some other algorithms which use graph representations to find clusters with different densities such as SNN [9]. This algorithm uses idea of \( k \) nearest neighbors. More precisely, SNN finds the \( k \) nearest points (adjacency set) for each point then it adds an edge between two points if adjacency set of the points have same points more than a threshold. After the graph is built, through some more measures, clusters are made.

3. THE PROPOSED CLUSTERING ALGORITHM

In this section, we propose a new Algorithm for Clustering Spatial Data with different densities (ACSD). This algorithm takes a parameter \( k \) and tries to create a corresponding graph in which the degree of each point is around \( k \). The graph has two important properties: (1) all points of a cluster lie in the same connected component, (2) two points from different clusters are disconnected. In other words, each connected component represents a cluster.

Looking to Fig. 1, it is obvious that there are two clusters. The central one is denser, while the other one is sparser. We can make a graph with mentioned properties and find clusters for points. This can be done by avoiding adding edges between the points in the border of central cluster and the points in the sparser cluster. In what follows, we try to formalize the idea of how to avoid adding edges between the borders, while edges between points in the clusters are...
The input of ACSD is a set of points in an n dimensional space $S$. Our approach works with any distance function that comes from different applications. However, for better visualization all the examples in the paper are represented for the 2-dimensional space with Euclidean Distance.

As discussed before, ACSD adds some edges between the points and assigns a level to each point. Let $adj(p)$ be the set of all points $q$ where there is an added edge between $p$ and $q$. Note that $adj(p)$ changes during iterations of the algorithm. level is an upper bound to indicate the maximum allowable distance between a point and its adjacent points. In fact, level should be chosen in such a way that border points of two near clusters become disconnected, while points in the same cluster get connected. In addition, we compute average distance of a point to its adjacent points; and use this number as additional information to decide whether or not two points should get connected.

**Definition 1.** level of a point: The level of a point $p$, denoted by $level(p)$ is a real number where $\forall q \in adj(p), \text{distance}(p,q) < level(p)$.

**Definition 2.** $AvgWgt$ of a point: The $AvgWgt$ of a point $p$, denoted by $AvgWgt(p)$ is the average distance of $p$ to all of its adjacent points.

ACSD runs in three phases. Here an outline for each phase is given and in the next three subsections the detailed explanations are discussed.

In the first phase, we assign preliminary level values to all points. Each point $p$ sends $2k$ feedbacks to its $2k$ nearest points. Sending the feedbacks means, $p$ declare its intention to become adjacent to them. When all points sent their feedbacks, each point will estimate its level according to the received feedbacks. Finally, we add some edges to the graph by considering the level values of pairs of points.

In the second phase, the level values are improved in multiple iterations. At the start of each iteration, all points send feedbacks to a number of nearest points where this number is a function of their level values. Then each point $p$ updates its level to a more accurate one taking into account the levels of points from which it has received feedbacks. Finally, a new graph is built and the level values are updated.

In the third phase, the final graph is constructed. More formally, we add an edge between two points if the distance between them is lower than level of each point and $AvgWgt$ of the points are similar.

After construction of the graph, border points are identified using some extra measurements. Finally, we use a Breadth-First Search (BFS) algorithm, and according to border and core points, we identify the connected components of the graph. Remember from the properties of the constructed graph each connected component is a cluster in the data set.

### 3.1 Building the Preliminary Graph

In this phase, we aim to guess the initial level values of all points. To this end, each point sends $mk$ feedbacks to its $mk$ nearest points, where $m \geq k$. Initial level values are calculated according to these feedbacks. Notice that, for a specific point $p$ it is better to estimate its level according to the received feedbacks (sent feedbacks of points which have $p$ as their $mk$ nearest neighbors) rather than its sent feedbacks. For example, point $b$ in Fig. 1 sends most of its feedbacks to the points in the denser cluster, while it lies in the sparser cluster. $m$ should be set in a way that leads to appropriate estimation of the distance to the $k^{th}$ point from each point $p$. High or low values of $m$ may lead to inaccurate results. We have set $m$ to 2 which has become from an intuitive notion that each point may hope that its sent feedbacks may be answered by probability of $50\%$ from other points, given that there is no initial assumption before this phase of the algorithm.

As we want to keep the degree of each point to be almost $k$; each point $p$ sorts its received feedbacks according to their distances from $p$, in ascending order. If $p$ receives greater than or equal to $k$ feedbacks, its level will be the distance from the $k^{th}$ point in the sorted array; otherwise, its level is the distance from the last point.

After estimating the level values, we build a preliminary graph by adding an edge between each pair of points, if their distance is lower than both of their level values. The preliminary graph and levels of the points for a simple data set is shown in Fig. 3(a).

### 3.2 Iterating on the Graph

If points are distributed in the surface of the cluster uniformly, then it is clear that levels of the points in the cluster are the same. Even when points are distributed with a Gaussian distribution, levels of two close points in a cluster are the same. To exploit this idea, when a point $p$ sends feedbacks to its neighbors, it also sends its own level value. Therefore, each point evaluates the level values received from the neighbors and it updates its level value according to the received levels. For example, the level of point $c$ in Fig. 3(a) is 1.4 but after receiving feedbacks, its level changed to 1.1.

Now, assume a point $p$ in the border of a dense cluster which is close to a sparser one. When $p$ goes over the re-
ceved feedbacks, it encounters a long jump in their levels. In other words, \( p \) receives feedbacks with two different sets of levels - the ones which are received from dense cluster and the ones which are received from the sparser one - see point \( a \) in Fig. 1. With this idea, \( p \) can find which feedbacks are from the points in its own cluster and will set its level related to them, not the feedbacks from the other clusters.

Remember that level of each point roughly reflects the density of the cluster the point resides. Notice that, considering the distances will not work, as two points might be too close to each other while they are in a sparse cluster. For instance, point \( c \) and \( d \) in Fig. 1 are too close to each other, if point \( c \) uses the distances of received feedbacks instead of levels, it encounters a long jump but it is not meaningful. In fact, it sees a long jump as the distance to point \( d \) is noticeably smaller than the other distances, while it does not mean it is from a different cluster.

In what follows, we first define a useful notation and then, give the detailed explanation of this phase.

**Definition 3.** \( \alpha \)-similarity: Two numbers \( x \) and \( y \) are \( \alpha \)-similar if and only if

\[
\begin{align*}
x &\leq \alpha \times y \quad \text{if} \quad y \leq x \\
y &\leq \alpha \times x \quad \text{if} \quad x < y
\end{align*}
\]

In this phase, graph is improved in multiple iterations. Each iteration consist of four major steps:

1. Each point sends at most \( k \) feedbacks according to its level. To be more precise, point \( p \) traces on the list of its \( 2k \) nearest points in ascending order of their distance to \( p \), and sends its feedback to each point \( q \) if distance \( (p, q) \) is less than levels of both points \( p \) and \( q \). It goes over the list to send at most \( k \) feedbacks or finishes the list.

2. Each point \( p \) finds a long jump in the levels of received feedbacks. In order to do so, it sorts the received levels and then finds the first point \( q \) whose level is not \( \alpha \)-similar to the next point’s (see Definition 3). Then \( p \) updates its level to the level of \( q \). Note that, the border points of a dense cluster will get the accurate level value which prevents adding edges between them and points of the sparser cluster. Here \( \alpha \) is a parameter of our algorithm. By increasing \( \alpha \), the algorithm finds less clusters while they are distinct and by decreasing \( \alpha \), it finds more but less distinct clusters. (All our experimental results have \( \alpha = 1.15 \), Fig. 3(b) shows levels of the points after Step 1 and Step 2).

3. We build a new graph in this step. There is an edge between two points \( p \) and \( q \) if and only if distance \( (p, q) \) is less than both their levels and their levels are similar. Fig. 3(b) shows the constructed graph for this data after this step.

4. In this step, each point updates its level once more. If it has more than \( k \) edges then its level is the distance to the \( k^{th} \) nearest point; otherwise, it equals to the distance to the last point. This update improves levels. Some points might get a large level value, while they are just connected to few close points. Therefore, their levels are high falsely; and as a result, we over cluster (See Fig. 3(b)). This happens when levels of some points which are adjacent to each other are high while size of their edges in the graph is low. This update also prevents over clustering for the future iterations. levels of the point after this step is shown in Fig. 3(c).

### 3.3 Constructing The Final Graph

In this phase, the final graph is constructed. All the levels and \( \text{AvgWgts} \) are set so far. As mentioned before, every connected component of this graph represents a cluster.

The final graph is built as follows. We add an edge between each pair of points \( p \) and \( q \) if distance \( (p, q) \) is less than the level of both of them and their \( \text{AvgWgts} \) are \( \alpha \)-similar (see definition 3). The distance constraint leads to a noticeable reduction in adding edges between two near clusters. The \( \text{AvgWgt} \) constraint guarantees that two points from different clusters with different densities share as few as possible edges even if they are too close to each other.

### 3.4 Clustering

Finally, we run an algorithm like DBSCAN to find the final clusters. Note that in the final graph there might be few edges between two different clusters which we will prune. We use this fact to find border and core points in the clusters. More precisely, for each point \( p \) let vector \( v_p \) be the vector which is started from \( p \) towards the origin. In addition, think of each edge from \( p \) as a vector started from \( p \) towards the neighboring point. Compute the angles between these vectors and \( v_p \), sort and store these angles in \( \text{p.angles} \). Now a point \( p \) is called a border point if there is a gap with size \( \theta \) in \( \text{p.angles} \) and a core point otherwise; where \( \theta \) is another parameter of ACSD. We use \( \theta = \frac{\pi}{6} \) for our benchmarks, \( \theta \) can be changed in order to allow or disallow the surface of the clusters to have obtuse angles.

After building the graph and detecting border and core points; the algorithm starts clustering like DBSCAN. To be more precise, the algorithm starts with a core point \( p \) and
Definition 4. Reachability: A point \( q \) is reachable from a core point \( p \) if there is a path \((p, q_1, q_2, \ldots, q_n, q)\) in the graph between \( p \) and \( q \) in which all \( q_i \)s are core points (\( q \) might be a border point).

3.5 Time Complexity of ACSD
In the first phase, each point finds \( 2k \) nearest points which takes \( O(n \log n + kn) \) time [8]. In the second phase, after some iterations all levels converge. As the maximum allowable number of iterations is a constant number (to be discussed in the next section), the time complexity of this phase is \( O(kn) \). In the last phase, to build the final graph we use \( O(kn) \) operations. Therefore, overall complexity is \( O(n \log n + kn) \). There is a fast approximation algorithm for finding the \( k \) nearest neighbors of a point in [13]. It is also shown that if points are in 2D or 3D the complexity is \( O(n \log k(k)) \)[7].

4. EXPERIMENTAL RESULTS
In order to evaluate ACSD, some experiments were conducted. ACSD is compared with SNN, DBSCAN and DD-DBSCAN which are density based clustering methods. In addition, SNN and DD-DBSCAN use the notion of different densities similar to ACSD and are selected for comparisons.

4.1 Data sets
The experiments are done on two groups of data sets: The Chameleon data sets [2] that are used to evaluate the capability of the algorithms to correctly identify clusters in noisy environments; and three synthetic data sets (data sets 1, 2 and 3) that are constructed to evaluate the capability of the algorithms to detect very close clusters with different densities [1]. In data set 1, there are 3000 points (1000 points in a \( 2 \times 2 \) square, 1000 points in a \( 4 \times 4 \) square and 500 points in a \( 6 \times 6 \) square). In data set 2, there are 5700 points (containing two uniformly distributed dense clusters and two sparser ones). Data set 3 and its specifications can be seen in Fig. 2.

4.2 Parameter Tuning
ACSD has three parameters: \( k \), \( \alpha \) and the number of iterations. \( \alpha \) and the number of iterations are set to 1.15 and 15, respectively. \( k \) is set as follows: high values of \( k \) helps identify the outlier and border points, because their AvgWgts and levels get higher. Therefore, these points will
be connected to few points in the graph constructing phase. In addition, by using higher values for \( k \), distinguishing between clusters with different densities will be easier. On the other hand, \( k \) should not be set excessively high, since the higher the value of \( k \) is, the more tendency ACSD have to merge clusters by creating more intra-cluster edges in the produced graph. This parameter is set to 20 in all experiments.

For SNN algorithm, its default values of \( k \) and \( \text{MinPts} \), i.e. \( k = 50 \) and \( \text{MinPts} = 34 \) are used. Its \( \epsilon \) parameter is also tuned to its best value for each data set. Finally, DD-DBSCAN is used with \( \text{Minpts} = 4 \) and \( \alpha = 0.2 \) as their default values and \( \epsilon \) is tuned to its most accurate value for each data set.

### 4.3 Results and Discussion

The results of the proposed algorithm, including the constructed graphs and the final clustering are presented in this subsection. In addition, the results are compared with SNN and DD-DBSCAN methods.

The final graphs of the synthetic data sets and one of the Chameleon data sets is shown in Fig. 4. It can be seen that in all cases, the graphs almost discriminate the clusters from each other. However, still there exists some edges between the clusters for some data sets that is handled in the clustering phase.

Fig. 5 shows comparison between ACSD, DD-DBSCAN and SNN over the synthetic data sets. In data set 1, ACSD determines clusters correctly, while DD-DBSCAN and SNN both over cluster and merge clusters together in the same time. Therefore, if we increase the value of \( \epsilon \) more clusters get merged together; and if this value is reduced, more over clustering happens. There may exist some average values for \( \epsilon \) that both over clustering and under clustering happen. In data set 2, similar results are obtained for SNN and DD-DBSCAN; therefore, they cannot cluster these data sets accurately as no value of \( \epsilon \) leads to perfect clustering. In addition, these algorithms do not recognize borders of the clusters correctly. Again in this data sets ACSD determines clusters perfectly and does not have any difficulty with border points of the clusters.

Finally, ACSD can accurately find all clusters in Chameleon data sets. Results of ACSD over three important ones are shown in Fig. 6, while it is known that DBSCAN and many other methods cannot detect these clusters accurately [14, 9]. DD-DBSCAN, however, can perfectly find all clusters for these data sets. In addition, SNN cannot cluster some of the data sets according to our experiments (e.g. t.8.8.k).

In future, ACSD can be compared with other algorithms and on data sets with higher dimensions; however, this requires more space which we skip for this manuscript.

### 5. CONCLUSIONS

In this paper we proposed ACSD which can detect clusters with different shapes, sizes and densities in a noisy environment with lots of outliers. ACSD can automatically determine the number of clusters and their densities even if they vary a lot. It uses the structure of the data set itself via sending and receiving feedbacks, to periodically guess and form the clusters. Here, each point has an upper bound which indicates to the farthest distance a point can make connection to the other points. This method is not very dependent to its parameters, in other words, with small changes in the value of a parameter the clustering result will not change significantly; however, they can be used to enforce special structures to the clusters. In addition, the algorithm assigns a number (level) to each point and it is independent of distance function. Finally, as this method uses a new approach it has the capacity to be improved and applied in other fields as well.

### 6. REFERENCES

[1] ACSD Data Sets. 
[author home page/research/ACSD. [Online; accessed 27-Sep-2012].


