Discovering Spatial Patterns Accurately with Effective Noise Removal

Yu Qian Kang Zhang
Department of Computer Science, The University of Texas at Dallas, Richardson, TX 75083-0688, USA
{yxq012100, kzhang}@utdallas.edu

ABSTRACT

Cluster analysis is a common approach to pattern discovery in spatial databases. While many clustering techniques have been developed, it is still challenging to discover implicit patterns accurately when the data set contains two kinds of noise or outliers: 1) domain-specific noise; 2) noise similar to true data on size, shape, or density. This paper presents a two-step strategy to solve the problem effectively: firstly, groups of data points are separated into different layers according to their sizes and densities; then a layered visualization is provided to the user to separate noise and true data intuitively. Such a strategy not only produces user-desired results but also separates noise and true data accurately. After noise removal, a hierarchical clustering is performed on remaining data to discover natural clusters. The experimental studies on both benchmark data sets and real images show very encouraging results.

Keywords

Data Mining, Clustering, Noise Removal, Pattern Discovery

1. INTRODUCTION

As a primary approach to spatial pattern discovery, cluster analysis has attracted considerable interest in recent years. While many clustering methods have been proposed to discover patterns of various shapes, densities, and sizes, the problem of noise removal, however, has not been fully addressed due to two facts observed in many spatial databases: first, noise could be domain-specific. Noise in one case may not be noise in another. Second, noise or outliers may be close to the true data in their positions, or have similar density or size, which makes traditional separation methods fail. Noise that satisfies either situation described above is referred to as being *ambiguous* in this paper. Figure 1 shows a benchmark data set used by CHAMELEON [8], where the thick horizontal line crossing "GEORGE" is ambiguous: it should be regarded as noise in area of letter recognition; but it could also be true data representing a style or decoration, depending on different applications. General definition of noise based on density or size cannot be applied in such a case. Besides, since the thick line not only overlaps the true data but also has a similar density, it poses a challenging problem for existing noise removing methods.

DMKD'04, June 13, 2004, Paris, France.

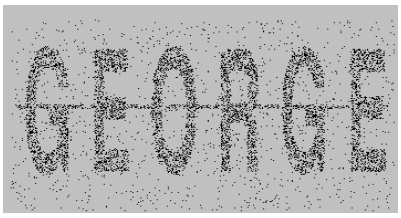


Figure 1. A benchmark data set that contains ambiguous noise.

This paper presents a novel approach to the discovery of implicit patterns in spatial data with ambiguous noise. The proposed approach, called CLEAN (CLustering by Eliminating Ambiguous Noise), removes noise before performing cluster analysis on the remaining noise-free data. We design such a procedure because: a) The presence of noise disturbs the clustering process, increases the complexity of clustering and error rate, b) many applications require a clustering algorithm to be capable of separating noise from true data, and c) noise removal can be a standalone means for cleaning any raw spatial data.

As illustrated in Figure 2, CLEAN consists of three stages:
1) Graph construction. The given data set is modeled with a *k*-mutual neighborhood graph.

2) Noise removal. This stage involves a two-step mechanism to guarantee effective handling of ambiguous noise: firstly, a fast graph partitioning method, *k*-core algorithm [12], is used to decompose the *k*-mutual neighborhood graph into small groups of data points, which are sorted into different layers according to their sizes and densities. Then a visualization of the data with the multi-layer structure is provided to the user. There are two modes provided to the user in the visualization: in basic mode, the user can select only one layer, which is called the *boundary layer*, to separate true data and noise. The layers above the boundary are regarded as true data and the layers below are regarded as noise; in advanced mode, the user can mark any layer as noise or true data through visual inspection and the remaining data will be the combination of the layers marked true. Basic mode can be run without user intervention: for data sets with similar properties, a learning approach is proposed to detect the boundary layer automatically.

3) Cluster analysis. After noise removal, a hierarchical clustering is performed on the remaining data to discover clusters of different shapes, sizes, and densities. In this stage, a compression [11] is applied on the remaining data to accelerate the hierarchical merging process.

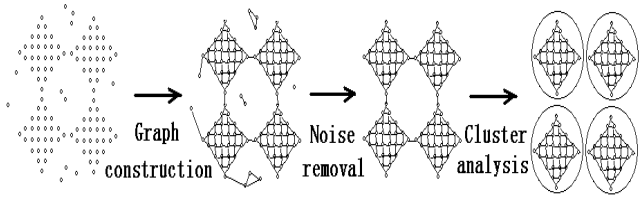


Figure 2. The three stages of CLEAN.

In summary, this paper's contributions include:

- Applying *k*-core algorithm to produce a layered structure of data points to remove noise more effectively than existing noise-handling approaches. The effective noise removal method reduces the complexity of the later steps significantly, and makes the hierarchical combination, the third step in Figure 2, error free. Besides, many clustering algorithms work well when the data set is "clean" while their performances degrade if the data set contains heavy noise. An effective noise removal method would enhance the effectiveness of such algorithms.
- A 3-D visualization of the layered structure of the data to allow user participation in handling ambiguous noise. The visualization avoids the parameter tuning of the *k*-core algorithm, and can also be used to assist similar clustering methods on parameter selection.
- A hierarchical clustering framework, which is able to discover clusters of arbitrary shapes, densities, and sizes efficiently. CLEAN is the key component of FAÇADE [10] and publicly available at our website: http://viscomp.utdallas.edu/FACADE, where one can run the CLEAN program with different input data sets and see the results intuitively.

The rest of this paper is organized as follows. The related work is discussed in Section 2. Section 3 introduces the data modeling method used in CLEAN. The *k*-core algorithm for noise removal is explained in Section 4. Section 5 describes our hierarchical clustering framework. Section 6 reports experimental results and presents a comparison between CLEAN and several representative clustering methods. Section 7 concludes the paper.

2. RELATED WORK

According to Han *et al.* [5], spatial clustering methods can be classified into four categories: partitioning method, hierarchical method, density-based method, and grid-based method. Since our main interest lies in noise removal for accurate pattern discovery with cluster analysis, this section focuses on the noise removing techniques used in existing clustering methods. From the perspective of noise handling, clustering methods can be classified into two categories: one category does not handle noise until clustering finishes. Most partitioning methods such as *k*-means [9] and many hierarchical methods belong to this category. The usual way to remove noise for the methods of this category is to judge the sizes of the produced clusters. If the size of a cluster is below a predefined threshold, it will be regarded as noise and removed from the final result. For example, RandomWalk [6] omits the small clusters whose sizes are below half of the average. Removing noise after clustering is a typical chicken and egg problem: to remove noise effectively requires an accurate clustering while the clustering methods usually suffer from the presence of noise. Besides, it is not easy to fix the threshold for different data sets. This category also includes some methods that do not remove noise from the final result. CHAMELEON [8] produces final clustering results containing all noise points, some of which are in separate clusters and others are mixed up with true data.

The other category aims at removing noise during the clustering process. Density-based approaches such as DBSCAN [4] and OPTICS [1] construct sophisticated similarity models with distance between data points and use two predefined thresholds, *Eps* and *MinPts*, to distinguish noise from true data. Although distance-based measurement can remove "salt and pepper" noise effectively, they may treat a sparse cluster as noise wrongly. SNN [3] is proposed to solve the problem by constructing a shared nearest neighbor graph on the given data set, where the degree of a node is decided by the number of its nearest neighbors instead of the distance between them. Thus both the sparse and the dense clusters can be discovered within the graph. The degree-based noise removing method used in SNN will be compared with CLEAN in Section 6.2.

Unlike the aforementioned approaches, CLEAN removes noise before the clustering begins. The noise removing process is standalone and can be applied to other algorithms. The noise removing method used in CLEAN, referred to as core-based noise removal, is not size-based or distance-based. Thus the sparse clusters will not be ignored. Through data visualization, CLEAN allows users to participate in the process of noise removal to detect ambiguous noise, which leads to an accurate pattern discovery. The following section will introduce the first step of CLEAN: using graph to model the spatial data set.

3. MODELING DATA WITH *k*-MUTUAL NEIGHBORHOOD GRAPH

Like many other hierarchical clustering algorithms, CLEAN starts with a graph representation of the given data. As noted by Harel and Koren [6], many clustering methods use sparse graph[1] model, which contains only a small subset of the edges of the complete graph, mostly those corresponding to higher similarity values. Limiting the number of the edges has two advantages: first, it reduces the time and space complexity. Second, the structure of a sparse graph reflects spatial proximity and expresses data distribution.

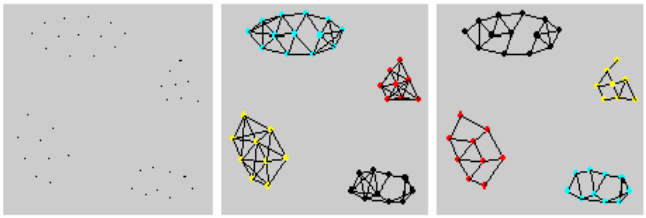
The commonly used sparse graph structures include: the *k-nearest neighborhood graph*, the *k-mutual neighborhood graph*, and the *Delaunay Triangulation* [7]. While the *Delaunay Triangulation* does not suit CLEAN, the first two structures will be discussed in this paper.

Generally, each vertex of the *k*-nearest neighborhood graph represents a data item. For each pair of data items, if either of them is among the *k*-most similar data items of the other, there exists an edge between the two corresponding vertices. In a spatial database, data items are points on a metric/dimensional space $\Re^d$ and the similarity of two data points is usually measured by the distance between them. The time needed to construct a *k*-nearest graph for *n* data points depends on the dimensionality of the

[1] A rough definition of sparse graphs is that the number of the edges in a sparse graph is much less than the square of the number of the vertices in it, i.e., $|E| << |V|^2$, for a graph *G (V, E)*, where $|V|$ is the number of vertices, $|E|$ is the number of edges.

underlying data set. Basically, it needs $O(nlogn)$ time for low-dimensional data sets and $O(n^2)$ for high-dimensional data sets.

Compared with a *k*-nearest neighborhood graph, for the same data set and *k*, a *k*-mutual neighborhood graph contains fewer edges. For each pair of data items, only if both of them are among the *k*-most similar data items of each other, can there be an edge between the two corresponding vertices. The computation efficiency of the *k*-mutual neighborhood graph is the same as that of the *k*-nearest neighborhood graph. Figure 3 shows a *4*-nearest neighborhood graph and a *4*-mutual neighborhood graph of a spatial data set.



(a) (b) (c)

Figure 3. (a) A spatial data set; (b) its *4*-nearest neighborhood graph; (c) its *4*-mutual neighborhood graph.

Generally, the advantages of representing data using *k*-nearest or *k*-mutual graph include: first, data points that are far apart are completely disconnected from the graph. Second, the constructed graph is able to represent the natural density dynamically. In a dense region, the neighborhood radius of a data point is determined narrowly, and in a sparse region, the neighborhood radius becomes wide. Third, the number of graph edges is linear to the number of vertices. The first two advantages decide the graph structure can be used to distinguish noise and true data.

4. HANDLING NOISE

Aiming at distributing data points into different layers, a graph-theoretic decomposition is based on the structural information, i.e., the connections of the graph nodes, as described in this section.

4.1 The *k*-core Algorithm

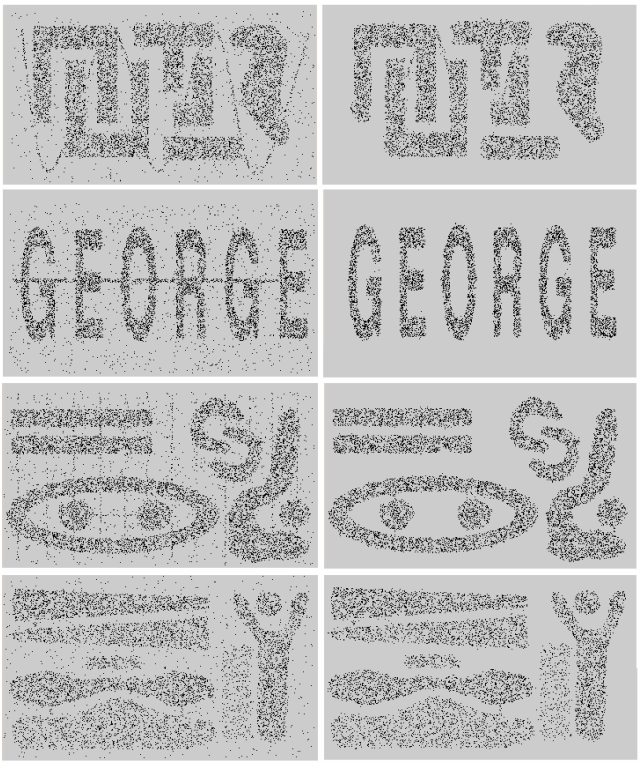
The notion of a core is introduced by Seidman [12]: let $G = (V, E)$ be a graph. V is the set of vertices and E is the set of edges. A subgraph $H_k = (W, E \mid W)$ induced by the set W is a *k-core* or a *core of order k* iff $\forall v$ in W: degree $(v) \geq k$ and H_k is the maximum subgraph with this property. The core of maximum order is also called the *main core*. The cores have the following properties:

- They are nested: $\forall i<j \rightarrow H_j \subseteq H_i$.
- There exists an efficient algorithm to determine the core hierarchy.
- A core is not necessarily a connected subgraph.

The algorithm for determining the core hierarchy is simple: from a given graph $G=(V, E)$, recursively delete all vertices of degrees less than k and lines incident with them, the remaining graph is the *k*-core. Known as the *k*-core algorithm, it costs only $O(m)$ time, where m is the number of edges for the given graph [12]. To avoid confusion, we will hereafter use k_c as the k used in *k*-core algorithm and k_m as the k used in *k*-mutual graph. For a *k*-mutual graph with n vertices and m edges, we have $m \leq k_m n/2$ if n is the number of vertices[2], so applying *k*-core algorithm to a *k*-mutual graph requires only linear time.

Batagelj *et al*. [2] use *k*-core algorithm as an efficient approach to the decomposition of large graphs and have verified its efficiency and effectiveness on graph partitioning. We propose to use core decomposition as an effective noise removing method for spatial data clustering. After obtaining the core hierarchy, noise removal becomes a simple process: remove the cores of orders smaller than k_c as noise and keep the cores of orders larger than or equal to k_c as true data.

It is not hard to justify why core decomposition suits spatial clustering. According to the definition of a core, each vertex of a core of k_c must have a degree greater than or equal to k_c even after all other cores with orders smaller than k_c have been removed, i.e., removing the cores of smaller orders will not decrease or affect the quality of the cores of bigger orders. While the connectivity of a *k*-mutual graph reveals the density distribution, such definition matches the natural rule that removing noise should not decrease or affect the density of true data. It is reasonable to assume the cores of small orders as noise and those of bigger orders as true data. The experiments on four benchmark spatial data sets of CHAMELEON [8], as illustrated in Figure 4, have verified our hypothesis: the core decomposition can produce surprisingly "clean" results.



(a) (b)

Figure 4. (a) The original data sets; (b) The resulting data sets after applying core decomposition.

[2] Given a data set, the corresponding *k*-nearest graph $G=(V, E)$ and the *k*-mutual graph $G'=(V', E')$, we have $|V|=|V'|$, $|E'|\leq |E|\leq k|V|$, and $|E|+|E'|=k|V|$, so $|E'|\leq k|V'|/2$.

So far there have been two parameters that would affect the result of noise removal: k_m and k_c. Our experiments reveal an important phenomenon: for a wide range of k_m., there exists a corresponding k_c so that the pair (k_m, k_c) removes noise effectively. For the four benchmark data sets shown in Figure 4, we let k_m be some values from 10 to 300, then choose k_c manually (k_c is chosen to remove most noise before losing true data) for each k_m. and check how much noise can be removed. As shown in Figure 5, when k_m is between *40* and *150*, more than *90%* noise can be removed before losing any true data; when k_m is between *20* and *200*, more than *70%* noise can be removed. Further experiments show that if the loss of an insignificant part of true data is allowed, the valid range of k_m can be much wider.

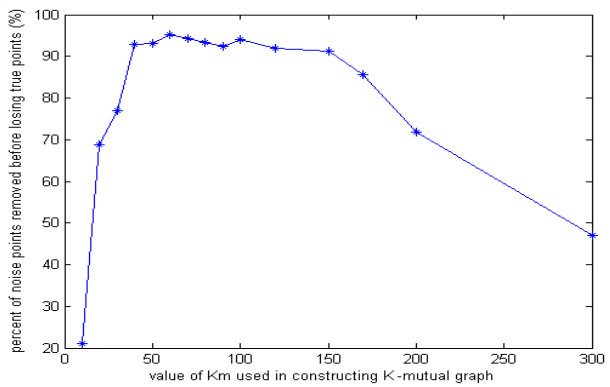


Figure 5. The ratio of noise removed for different k_m.

While k_m can be chosen more freely, it is difficult to automatically and correctly select the correspondingly k_c without domain knowledge. This is because of the lack of a universal noise definition. While k_c can be regarded as a threshold to indicate when to stop the noise removing process, without defining noise properly, it is impossible to decide which data point is noise and should be removed. Unfortunately, noise is usually a domain-specific concept. Noise in one case may not be noise in another. Since generally defining what is a noise is beyond the scope of this paper and involving domain knowledge becomes inevitable to recognize ambiguous noise, here we provide two alternatives: let the user specify k_c through a visualization of the data with multi-layers where each layer represents the data points belong to the same core. Thus the user can specify how much noise he/she wants to remove by choosing which layer represents k_c. Compared with the similar parameters like the *Eps* and *MinPts* used in DBSCAN [4] and SNN [3] and their selections, such a visualization used in CLEAN can determine the parameter more intuitively and conveniently. The other approach is to define approximately the density-based noise, which can help to produce a default k_c as a suggestion to a non-expert user for automatic noise removal. The two approaches will be addressed in Section 4.2 and 4.3, respectively.

4.2 Choosing k_c through Data Visualization

Figure 6 is the 3-D visualization of the layered structure of the data after applying the *k*-core algorithm. Each data point is assigned a core id, which is used as the third coordinate. Points of different cores are represented with different gray levels in Figure 6. Thus the data set is separated into many layers and each layer contains the data points of the same core. While different cores mean different densities or sizes, the data set is "segmented" into many small groups. Users can judge which groups are noise easily through visual inspection. As shown in Figure 6, the thick line crossing "GEORGE" has been separated into a layer far from true data and the 3-D visualization helps the user select the correct k_c to remove it easily. There are two modes provided to the user through the visualization. The basic mode accepts a user input about the boundary layer, i.e., k_c, and then the layers above k_c will be regarded as true data and kept while other layers will be removed as noise. In advanced mode, the system can accept a set of user inputs to specify which layers are true data or noise. Thus the noise removal becomes a customizable process. For example, in Figure 6, the user can keep the thick line by setting its layer as true, or the user can even remove the right foot of the letter "R" if he/she thinks the correct letter should be "GEOPGE". The visualization provides a full support for user participation in noise removal.

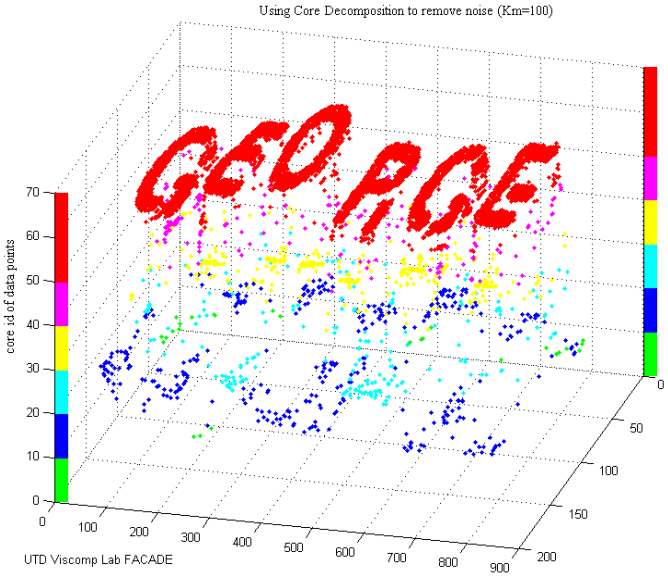


Figure 6. The visualization of the core hierarchy.

4.3 Separating Noise from True Data Automatically

The above sections have shown the ability of core decomposition in separating the true data from noise. One may wonder if this process can be completed automatically, i.e., use computation to decide which cores should be removed and which should be kept as true data. This section will provide an approximate definition of density-based noise. Based on this noise definition and several justified observations, we can compute the boundary layer between noise and true data, i.e., k_c through a learning approach.

Given a dataset *D*, suppose the corresponding *k*-mutual graph is $G=(V, E)$, $|V|=n$ and $|E|=m$. Apply the *k*-core algorithm to *G*, and obtain the set of cores, denoted by H_0, H_1, ..., H_{x-1}, H_x, where H_i represents the core of order *i*. Let $S_x = H_x$, $S_{x-1} = H_{x-1} - H_x$,..., $S_1 = H_1 - H_2$, $S_0 = H_0 - H_1$, and S_x is the main core, as illustrated in Figure 7. For the convenience of presentation, we call S_k a *proper core* since it includes the data points that are of the same connectivity level while H_k includes both S_k and all other data points that have a greater connectivity. The *size* of S_k, denoted by $|S_k|$, is the number of data points of S_k. We have $|S_0|+|S_1|+...+|S_x| = n$ and there are $x+1$ proper cores in total.

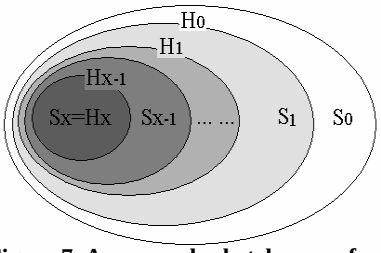


Figure 7. An example sketch map of cores.

Being the main core, S_x must be the densest part of the graph G and contain at least $x+1$ data points. The following observation can be regarded as our basic assumption and starting point.

Observation 0: The data points of S_x are true data.

Observation 0 has been verified in all of our experiments. It is a reasonable assumption since if a large and the densest part of a graph were not true data, no other part should be regarded as true data without domain specification.

Observation 0 defines the first part of true data. Then we are able to provide an approximate definition of density-based noise:

Definition 4.1 (Density-based Noise):
Density-based Noise is neither: 1) closely related to true data; nor 2) of big size and dense.

The definition of density-based noise can be explained intuitively: if a set of data points is closely related to the true ones, they should not be separated from the true points; and if a large set of data points is dense, without domain specification, they cannot be noise either. From this point on, the paper will consider noise being determined by density.

There are two important observations that follow the definition of noise, both being verified by our experiments:

Observation 1: Let $p \in S_i$ and $q \in S_j$ denote two arbitrary data points in two proper cores, if p is noise and q is true data, then $i \neq j$.

This observation implies that noise and true data do not co-exist in a proper core, which can be justified as follows: since the data points in a proper core have similar density, if some points in it are true data, it is unreasonable to regard others as noise. Thus noise and the true data must be in different proper cores. For the convenience of presentation, we will simply classify proper cores into noise cores and true data cores. In-depth observation on the indices of cores further reveals the location of noise: all noise cores have smaller indices than the true data cores.

Observation 2: $\forall i, j$, if S_i is noise and S_j is true data, then $i<j$.
Observation 2 can be easily justified using Observation 1 and Definition 4.1.

Definition 4.1 and the three observations simplify the automatic noise-removing problem into finding the k_c so that all the proper cores with orders smaller than k_c will be removed as noise. Suppose core H_t being true data (Starting from $t=x$, i.e., Observation 0), we check if S_{t-1} is true data or noise. According to Definition 4.1, we first judge if S_{t-1} is closely related to H_t, if so, S_{t-1} is a true core; otherwise, we continue to check if S_{t-1} is large and dense enough, if so, S_{t-1} is a true core; otherwise, S_{t-1} is a noise core and we can output t as k_c according to Observation 2. Figure 8 depicts this algorithm. In Figure 8, *RC*, *RS*, and *RD* are relative closeness, relative size, and relative density, respectively. Since the k-mutual graph uses connection to represent similarity, the closeness between H_t and S_{t-1} can be measured using the number of edges between them: according to the definition of core, each vertex of S_{t-1} has at most $t-1$ edges connected to the vertices of H_t, so the maximum edge number between H_t and S_{t-1} is $|S_{t-1}|(t-1)$. Let E_i denote the number of edges in S_i and E_inter_i the number of edges between H_{i+1} and S_i. The closeness between S_i and H_{i+1} and the relative closeness of the x pairs of core and proper core are defined as follows:

Definition 4.2 (Closeness and Relative Closeness)
The *closeness* between S_i and H_{i+1} is: $C_i=E\_inter_i/(i|S_i|)$ and the *relative closeness* of the x pairs of core and proper core is defined as: $RC=(\sum_{i=0}^{x-1}|C_i|/x)P_c$, where P_c is a threshold on closeness.

```
Algorithm Find-k_c
begin
  For each proper core S_i, compute |S_i|, |C_i|, |D_i|, and
RS, RC, RD for the x+1 proper cores;
  i=x;
  While (i≥0)
    if (|C_i|>RC)
        continue;
    else
      if (|S_i|>RS) and (|D_i |>RD)
          continue;
   else    return i;
  i=i-1;
end
```

Figure 8. The algorithm for finding k_c

Similarly, we can define the relative size and the relative density for the $x+1$ proper cores.

Definition 4.3 (Relative Size)
The *relative size* for $x+1$ proper cores is defined as: $RS=(\sum_{i=0}^{x}|S_i|/(x+1))P_s$, where P_s is a threshold on size.

Definition 4.4 (Density and Relative Density)
The *density* of S_i is: $D_i=|E_i|/|S_i|$ and the *relative density* of the $x+1$ proper cores is defined as:
$RD=(MAX(D_i)-MIN(D_i))P_d+MIN(D_i),\ 0\le i\le x$, where P_d is a threshold on density.

P_c, P_s, and P_d are obtained through an independent learning algorithm from training data sets of similar sizes and types of noise. Definitions 4.1, 4.2, 4.3, and 4.4 help the learning algorithm differentiate the noise from the true data effectively. Section 6.3 will evaluate the automatic detection of the boundary layer. In conclusion, we have Theorem 4.1:

Theorem 4.1. Algorithm Find-k_c identifies the boundary of noise cores and true cores.
Proof. According to Definitions 4.1, 4.2, 4.3, 4.4, Observations 0, 1, 2, and the induction of algorithm Find-k_c, all the cores with

orders smaller than k_c, i.e., the output of Find-k_c, are density-based noise while other cores are true data. □

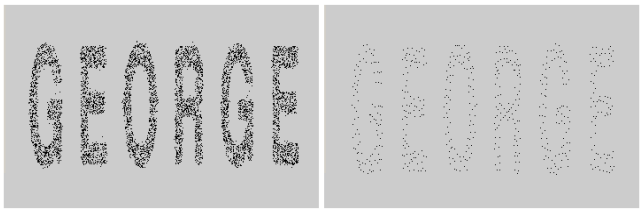
Although the *k*-core algorithm removes noise effectively, it cannot discover the boundaries of natural clusters. A natural cluster may have regions of different densities, and thus consist of cores of different orders. On the other hand, different natural clusters may have similar densities, and thus share one core. The following sections will introduce how to discover, from the result of *k*-core algorithm, the natural clusters of different shapes, densities, and sizes.

5. HIEARCHICAL CLUSTERING

As shown in Figure 4, the data set after applying the *k*-core algorithm is nearly noise-free, which provides an opportunity for many clustering approaches to work effectively. This section will present a hierarchical clustering process that can discover arbitrarily shaped clusters efficiently.

5.1 Compression and Sub-Graph Construction before Clustering

We apply GraphZip [11] to compress the remaining data to improve the efficiency of hierarchical combination. Each data point in the compressed data set represents a group of data points in the original data set. Figure 9 demonstrates the effect of GraphZip: the number of data points is greatly reduced while the spatial pattern is preserved.



(a) **(b)**

Figure 9. The data set (a) before (b) after applying GraphZip.

After compression, the next step is to prepare the initial groups for hierarchical combination. In CLEAN, the data points represented by a single data point after compression form an initial group. The chief requirement on preparing initial groups is that the points of different natural clusters should not be in the same initial group. Figure 9 shows that this requirement is satisfied. Thus we can begin hierarchical merging.

5.2 Merging Criterion

The basic idea of hierarchical merging is as follows: we continuously choose the most appropriate pair, from the initial groups, to merge until reaching one cluster. At each hierarchical level a value M is computed for each pair of groups, denoted by $M(i,j)$ for groups i and j. The hierarchical process merges the pair of groups that maximizes M at each hierarchical level until all the groups have been merged, and there is a combination criterion to specify how to compute M. As the hierarchical process continues, the number of groups decreases by *1* at each hierarchical level. In CLEAN, the value M is computed based on the *k*-mutual graph constructed on the original graph in the first step. Each data point of the initial groups has its corresponding vertex in the *k*-mutual graph. Suppose the *k*–mutual graph is $G=(V, E)$ and $S_1, S_2, \ldots, S_t$ are sets of vertices corresponding to t initial groups, we denote the number of edges between S_i and S_j as $E(i, j)$, and the size of S_i is defined as the number of vertices in S_i, denoted by $|S_i|$. The combination criterion is defined as follows:

$$M(i,j)=E(i,j)^2/MIN(|S_i|,|S_j|) \quad (1)$$

Formula (1) favors the number of connections between two groups over their sizes. The more connections, the more likely the two groups will be merged. On the other hand, if the connection is the same for two pairs of groups, formula (1) will merge the pair containing the smallest group first. Formula (1) favors adding points to a big group as long as the number of the points being added is small enough. Thus formula (1) can add small groups of points to the clusters continuously. According to the definition of formula (1), small groups will be merged into big groups in an order according to the number of connections. Using formula (1) will merge two natural clusters only after all of their sub-clusters have been merged.

6. PEROFORMANCE EVALUATION

The first part of this section will compare CLEAN with CHAMELEON [8], a well-know graph-based hierarchical spatial clustering algorithm. Then we will apply the automatic version of CLEAN to some real images on INTERNET and evaluate the automatic version of CLEAN.

6.1 Comparison with CHAMELEON on Cluster Quality

The final clustering results of the four data sets in Figure 4 (a) are illustrated in Figure 10. Each produced cluster has its own gray level. Figure 10(a) shows the clustering results of CHAMELEON[3] while Figure 10 (b) is the results of CLEAN. Since CHAMELEON experimentally outperformed previous systems on cluster quality, this section compares the experimental results of CLEAN with CHAMELEON only. Although the results in Figure 10(b) appear very similar to those in Figure 10 (a), CLEAN runs much faster than CHAMELEON, requires less user-supplied parameters, and most importantly, is able to remove noise.

6.2 Comparison with Degree-based Noise Removal

Degree-based noise removal is proposed in SNN [3]. We download the source code of SNN from [13] and compare its noise removal result with CLEAN, as shown in Figure 11. Figure 11 shows that the ambiguous noise is not completely removed by SNN. Besides, with the support of the visualization, the parameter selection of CLEAN is much easier than that of SNN.

Figure 11. Noise removal with (a) SNN; (b) CLEAN

[3] The second data set has not been used in CHAMELEON paper so no result shows.

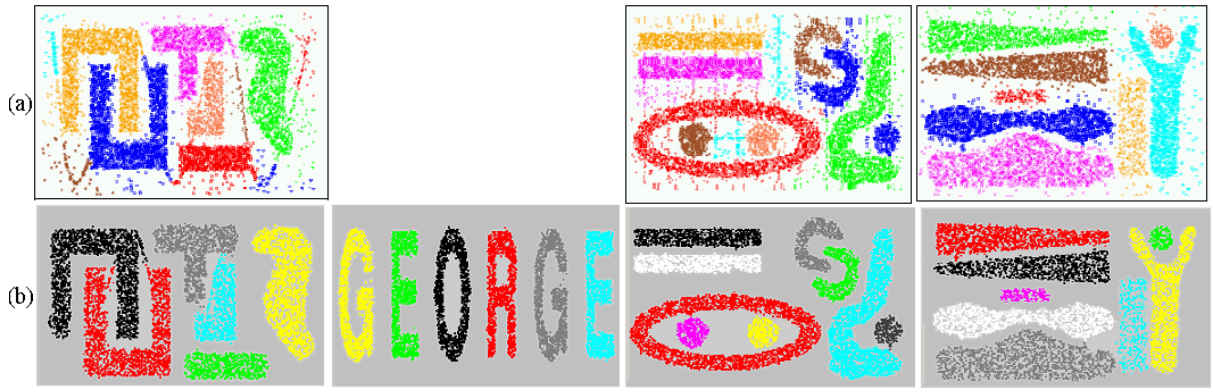


Figure. 10. The clustering results of (a) CHAMELEON; (b) CLEAN;

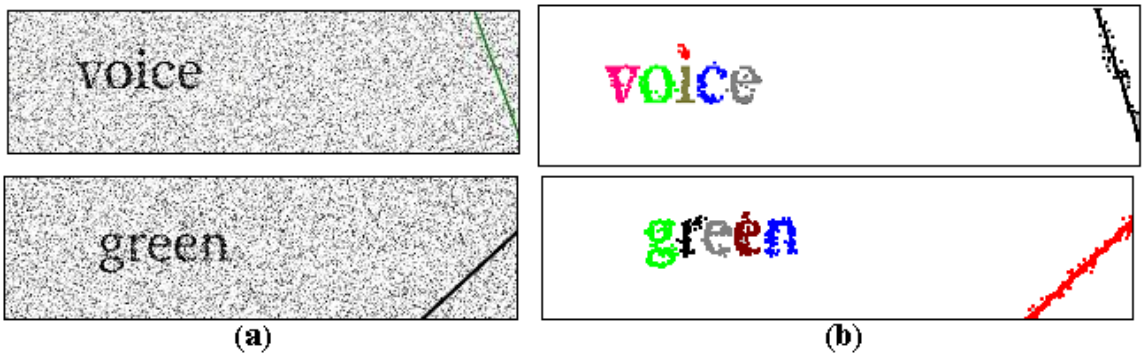


Figure 12. (a) The original images; (b) the clustering results using CLEAN

6.3 Clustering Real Images

This section will evaluate the automatic version of CLEAN, i.e., using the learning approach to decide the boundary layer automatically instead of user intervention. Figure 12. (a) shows two security images used by Yahoo to prevent automatic form filling in application of email address. Yahoo requires the user to recognize the letters by visual inspection and fill them into the application form so that programs for auto-filling fail. This means such images may be difficult to be recognized by computer programs. We collect a series of such images from Yahoo website and all of them have similar properties on size and noise ratio, so it is especially suitable for CLEAN to learn the P_c, P_s, and P_d by training, as described in Section 4.3. After training, the values of P_c, P_s, and P_d are fixed at 0.2, 0.5, and 0.9, respectively, and we fix the k_m at 20 due to the images have similar sizes. Figure 12 (b) demonstrates the results after performing trained CLEAN on two testing data sets. Each letter is clearly shown and could be recognized by a matching algorithm followed. The experiment evaluates the ability of CLEAN on discovering patterns in similar data sets automatically.

Now let us analyze the time complexity of CLEAN. The first step of CLEAN is to model the given data set with a *k*-mutual graph, which needs *O(nlogn)* time for 2-D data sets; Step 2 is core decomposition, both *k*-core algorithm and *Find-*k_c algorithm can be completed in *O(n)* time since the number of edges of a *k*-mutual graph is linear to the number of vertices; In Step 3, GraphZip requires *O(nlogn)* time to compress *n* data points into $\sqrt{n}$ [11], then the hierarchical merging of the $\sqrt{n}$ initial groups costs *O(n)* time. In summary, all steps can be completed in *O(nlogn)* time. On the space complexity, CLEAN requires *O(n)* space to compute and store the corresponding graph of the whole data set.

Table 1 compares CLEAN with three representative clustering algorithms with typical measurements. Among the three representatives, RandomWalk [6] and CHAMELEON are typical graph-theoretic hierarchical approaches, and SNN is the latest published progress. Table 1 shows that CLEAN substantially outperforms the three representative algorithms on at least one of three aspects: scalability, parameter minimization, and noise handling.

7. CONCLUSION

This paper has described a novel hierarchical graph-theoretic clustering approach, CLEAN, which can remove ambiguous noise and discover clusters of different shapes, densities, and sizes accurately. The paper has shown two important results: firstly, the data points can be separated into different layers using the *k*-core algorithm. The data set can be partitioned into many small groups with different densities or sizes. Thus noise and true data are clearly separated. Secondly, the visualization of the data with a layered structure allows a customizable noise removal. Users can "assemble" the remaining data according to domain requirements. Future work will focus on three issues: the detection of the terminating point of the hierarchical merging process, i.e., to estimate the number of clusters of the given data set; the application of CLEAN on image segmentation and pattern recognition, two common applications of spatial clustering algorithms; the effectiveness of combining *k*-core algorithm and the corresponding visualization with existing clustering methods for accurate noise removal.

8. REFERENCES

[1] Ankerst, M., Breunig, M., Kriegel, H. P., and Sander, J. (1999). OPTICS: Ordering Points To Identify the Clustering Structure. *Proc. 1999 ACM-SIGMOD Conf. on Management of Data (SIGMOD'99)*, pp. 49-60.

[2] Batagelj, V., Mrvar, A., and Zaversnik, M. (2000). Partitioning approaches to clustering in graphs, *Proc. Graph Drawing'1999,* LNCS, pp. 90-97.

[3] Ertoz, L., Steinbach, M., and Kumar, V. (2003), Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data, In *Proc. of SIAM DM'03*.

[4] Ester, M., Kriegel, H. P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise, *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press, pp. 226-231.

[5] Han, J., Kamber, M., and Tung, A. K. H. (2001). Spatial clustering methods in data mining: A survey, H. Miller and J. Han (eds.), *Geographic Data Mining and Knowledge Discovery,* Taylor and Francis.

[6] Harel, D. and Koren, Y. (2001). Clustering spatial data using random walks, *Proc. 7th Int'l Conf. Knowledge Discovery and Data Mining (KDD-2001)*, ACM Press, New York, pp. 281-286.

[7] Jain, A. K., and Dubes, R. C. (1988). *Algorithms for Clustering Data,* Prentice-Hall advanced reference series. Prentice-Hall, Inc., Upper Saddle River, NJ.

[8] Karypis, G., Han, E., and Kumar, V. (1999). CHAMELEON, A hierarchical clustering algorithm using dynamic modeling, *IEEE Computer*, Vol.32, pp. 68-75.

[9] McQueen, J. (1967). Some methods for classification and analysis of multivariate observations, *Proc. of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281-297.

[10] Qian, Y., Zhang, G., and Zhang, K. (2004) FACADE: A Fast and Effective Approach to the Discovery of Dense Clusters in Noisy Spatial Data, In *Proc. ACM SIGMOD 2004 Conference*, Paris, France, 13-18 June 2004, ACM Press. (Demo Abstract)

[11] Qian, Y. and Zhang, K. (2004) GraphZip: a fast and automatic compression method for spatial data clustering. In *Proc. of the 2004 ACM Symposium on Applied Computing (SAC'04)*, pp. 571-575.

[12] Seidman, S. B. (1983). Network structure and minimum degree. *Social Networks*, 5, pp. 269-287.

[13] http://www.cs.umn.edu/~ertoz/snn/

Table 1. The comparison between CLEAN and three representative clustering approaches

	Running Time (for n data points and m initial groups)	Finding clusters of different shapes?	Minimal input parameters		Robust to noise	
			Parameters used	How to set parameter values?	Robust?	Noise Removed?
CHAMELEON	nm+nlogn+ m*m*logm	Yes	MinSize, α, k	Fixed/Trial and error	Yes	No
Random Walk	nlogn	Yes	CE, NS, and weight thresholds	Fixed/Trial and error	Yes	Yes
SNN	n*n	Yes	k, MinPts, Eps	Fixed/Trial and error	Yes	Yes
CLEAN	nlogn	Yes	Km, Kc	Learned/Visualized	Yes	Yes