A Shrinking-Based Approach for Multi-Dimensional Data Analysis

Yong Shi, Yuqing Song and Aidong Zhang
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260
{yongshi, ys2, azhang}@cse.buffalo.edu

Abstract

Existing data analysis techniques have difficulty in handling multi-dimensional data. In this paper, we first present a novel data preprocessing technique called *shrinking* which optimizes the inner structure of data inspired by the Newton's Universal Law of Gravitation[22] in the real world. This data reorganization concept can be applied in many fields such as pattern recognition, data clustering and signal processing. Then, as an important application of the data shrinking preprocessing, we propose a shrinking-based approach for multi-dimensional data analysis which consists of three steps: data shrinking, cluster detection, and cluster evaluation and selection. The process of data shrinking moves data points along the direction of the density gradient, thus generating condensed, widely-separated clusters. Following data shrinking, clusters are detected by finding the connected components of dense cells. The data-shrinking and cluster-detection steps are conducted on a sequence of grids with different cell sizes. The clusters detected at these scales are compared by a cluster-wise evaluation measurement, and the best clusters are selected as the final result. The experimental results show that this approach can effectively and efficiently detect clusters in both low- and high-dimensional spaces.

1 Introduction

With the advance of modern technology, the generation of multi-dimensional data has proceeded at an explosive rate in many disciplines. *Data preprocessing* procedures can greatly benefit the utilization and exploration of real data. In this paper, we first present a novel data preprocessing technique called *shrinking*; then, as an important application of the data shrinking preprocessing, we propose a shrinking-based approach for multi-dimensional data analysis.

1.1 Related work

Commonly used as a preliminary data mining practice, data preprocessing transforms the data into a format that will be more easily and effectively processed for the purpose of the users. There are a number of data preprocessing techniques[21, 8]: data cleaning, data integration, data transformation and data reduction.

The need to cluster large quantities of multi-dimensional data is widely recognized. Cluster analysis is used to identify homogeneous and well-separated groups of objects in databases. It plays an important role in many fields of business and science. Existing clustering algorithms can be broadly classified into four types [14]: partitioning [13, 15, 20], hierarchical [27, 10, 11], grid-based [25, 24, 2], and density-based [9, 12, 4] algorithms. Partitioning algorithms start with an initial partition and then use an iterative control strategy to optimize the quality of the clustering results by moving objects from one group to another. Hierarchical algorithms create a hierarchical decomposition of the given data set of data objects. Grid-based algorithms quantize the space into a finite number of grids and perform all operations on this quantized space. Density-based approaches are designed to discover clusters of arbitrary shapes. These approaches hold that, for each point within a cluster, the neighborhood of a given radius must exceed a defined threshold.

Each of the existing clustering algorithms has both advantages and disadvantages. The most common problem is rapid degeneration of performance with increasing dimensions [12], particularly with approaches originally designed for low-dimensional data. To solve the high-dimensional clustering problem, dimension reduction methods [2, 1, 23]

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

have been proposed which assume that clusters are located in a low-dimensional subspace. However, this assumption does not hold for many real-world data sets. The difficulty of high-dimensional clustering is primarily due to the following characteristics of high-dimensional data:

1. High-dimensional data often contain a large amount of noise (outliers). The existence of noise results in clusters which are not well-separated and degrades the effectiveness of the clustering algorithms.
2. Clusters in high-dimensional spaces are commonly of various densities. Grid-based or density-based algorithms therefore have difficulty choosing a proper cell size or neighborhood radius which can find all clusters.
3. Clusters in high-dimensional spaces rarely have well-defined shapes, and some algorithms assume clusters of certain shapes.
4. The effectiveness of grid-based approaches suffer when data points are clustered around a vertex of the grid and are separated in different cells, as shown in Figure 1. In the d-dimensional space $\mathbb{R}^d$, there may be 2^d points distributed in this manner. The cluster formed by these points will be ignored because each of the cells covering the cluster is sparse.

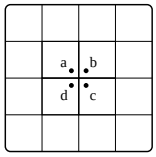


Figure 1: Points a, b, c, and d are located near a vertex of the grid and are separated in four neighboring cells. The four neighboring cells contain no other points.

In particular, there are several algorithms which are related to the data analysis method we will present in this paper as an application of the data shrinking preprocessing. However, each of them differs from our algorithm in a certain way. DENCLUE[12] concentrates on local maxima of density functions called *density-attractors* and uses a flavor of gradient hill-climbing technique for finding them. Cowen etc.[16] applied a randomized non-linear projections to uncover high-dimensional structure, preserving cluster separability. CURE[10] represents each cluster by a certain fixed number of points that are generated by selecting well scattered points from the cluster and then *shrinking* them toward the center of the cluster by a specified fraction. The *moving* concept in our data analysis method is different from some well known algorithms such as K-Means and SOM. Both the centroids of the clusters in K-Means and the nodes of the low-dimensional layout in SOM can be regarded as having some moving actions iteratively. On the other hand, the movement of our preprocessing concept is based on each data point instead of on only a few "representative" ones.

Many approaches [6, 17, 18, 19] have been proposed for evaluating the results of a clustering algorithm. M. Halkidi et al. [17] presented a clustering validity procedure which defines a validity index containing the information of the average degree of scatter within clusters and the average number of points between the clusters. C.F. Chen et al. [6] introduced a fuzzy validity function to measure the overall average compactness and separation of the fuzzy partition. These clustering validity measurements evaluate clustering algorithms by measuring the overall quality of the clusters. However, each clustering algorithm has its advantages and disadvantages. For a data set with clusters of various size, density, or shape, different clustering algorithms are best suited to detecting clusters of different types in the data set. No single approach combines the advantages of these various clustering algorithms while avoiding their disadvantages.

1.2 Proposed approach

In this paper, we first present a novel data preprocessing technique which optimizes the inner structure of data by simulating the Newton's Universal Law of Gravitation[22] in the real world. This data reorganization concept can be applied in many fields such as pattern recognition, data clustering and signal processing.

Then, as an important application of the data shrinking preprocessing, we propose a shrinking-based approach for multi-dimensional data analysis to address the inadequacies of current clustering algorithms in handling multi-dimensional data. This clustering method is combined with a cluster-wise evaluation measurement to select the best clusters detected at different scales.

The proposed algorithm consists of three steps which are data shrinking, cluster detection, and cluster evaluation and selection. In the data-shrinking step, data points move along the direction of the density gradient simulating the Newton's Universal Law of Gravitation, leading to clusters which are condensed and widely-separated. Following data shrinking, clusters are detected by finding the connected components of dense cells. The data-shrinking and cluster-detection steps are grid-based. Instead of choosing a grid with a fixed cell size, we use a sequence of grids of different cell sizes. Our technique also proposes a method to avoid the problem caused by points clustered near a vertex of a grid and separated in different cells, as shown in Figure 1. For each cell size, the processes of data shrinking and cluster detection are performed on two interleaved grids. Then, in the cluster evaluation and selection step, we evaluate clusters detected at different scales via a cluster-wise evaluation measurement and select the best clusters as the final result.

This paper offers the following primary contributions:

- We present a novel data preprocessing technique which optimizes the inner structure of data.
- We propose a data-shrinking process as an important implementation of the data preprocessing technique. It yields clusters which are condensed and well-separated. This data-shrinking steps can be used

as a preprocessing procedure for any cluster-detection algorithm. We will demonstrate how it will improve the performance of existing clustering algorithms in the experimental part.

- After the data-shrinking process, clusters are detected on the basis of density of cells. The algorithm is noise-insensitive and can detect clusters of any shape.
- Clusters are detected at different scales. The proposed multi-scale gridding scheme avoids the problem of determining a proper cell size and offers advantages for handling data sets with clusters of various densities.
- We propose a cluster-wise evaluation measurement to compare clusters at different scales and select the best as the final result. This approach can be used to unify multiple clustering algorithms, exploiting their advantages and avoiding their disadvantages.

The remainder of this paper is organized as follows. Section 2 introduces the concept of data shrinking preprocessing. From Section 3 on, we present the application of shrinking preprocessing to multi-dimensional data analysis. Section 3 introduces methods for the selection of multiscale grids for use in data shrinking and cluster detection. Section 4 discusses the data-shrinking process. In Section 5, a simple grid-based cluster detection method is presented. In Section 6, we discuss our definition of compactness as pertains to evaluating and selecting clusters. Section 7 presents experimental results, and concluding remarks are offered in Section 8.

2 Data shrinking preprocessing

We present a novel data preprocessing technique which optimizes the inner structure of data by simulating the Newton's Universal Law of Gravitation[22] which states that any two objects exert a gravitational force of attraction on each other. The direction of the force is along the line joining the objects. The magnitude of the force is proportional to the product of the gravitational masses of the objects, and inversely proportional to the square of the distance between them:

$$F_g = G\frac{m_1 m_2}{r^2}, \tag{1}$$

where F_g is the gravitational force, m1 and m2 are the masses of the two objects, r is the separation between the objects, and G is the universal gravitational constant.

Our data shrinking preprocessing computes a simulated movement of each data point in a dataset that reflects its "attraction" to neighboring data points. The degree of attraction is inversely proportional to the distance between points. This kind of data movement makes data points in the original dataset properly move to the center of gravity of the data group it belongs to. In this way the densities of the data groups are increased, and the outliers are further isolated. We can also refer to the concept of infiltration mechanism[26] in which materials such as water move from denser area to sparser one whereas in our case, the data point will move to the denser area nearby. Those data points which are far away should basically have no effect on the target data point and can be ignored. By only aggregating the gravitation (or effect) surrounding the target data point, proper direction and distance the target data point should move along can be acquired.

This data reorganization concept can be applied in many fields such as pattern recognition, data clustering and signal processing to facilitate a large amount of data analysis catetories.

3 Application of shrinking preprocessing to multi-dimensional data analysis

To demonstrate the advantages of the data shrinking preprocessing, we applied it to the multi-dimensional clustering problem which plays an important role in many fields of business and science. We propose a grid-based approach to data shrinking and cluster detection.

Choosing grids: Grid-based clustering methods depend heavily on the proper selection of grid-cell size. Without prior knowledge of the structure of an input data set, proper grid-cell size selection is problematical. We propose a multiscale gridding technique to address this problem. Instead of choosing a grid with a fixed cell size, we use a sequence of grids of different cell sizes. Data shrinking and cluster detection are conducted on these grids, the detected clusters compared, and those clusters with the best quality are selected as the final result.

Throughout this paper, we assume that the input data set X is

$$X = \{\vec{X}_1, \vec{X}_2, ..., \vec{X}_n\},$$

which is normalized to be within the hypercube $[0,1)^d \subset \mathbb{R}^d$.

One straightforward solution for the acquirement of multiscales is that we use a sequence of grids of exponentially increasing cell sizes. Let S_{min} and S_{max} be the minimal and maximal side lengths of grid cells, respectively. Let E_g be the factor used for increasing cell sizes. Then the side lengths of cells for the grids are, respectively,

$$S_{min},\ S_{min}*E_g,\ ...,\ S_{min}*(E_g)^{\eta} = S_{max},\ for\ some\ \eta \in \mathbb{N}. \tag{3}$$

The minimal side length of grid cells S_{min} depends on the *granularity* of the data, which is determined by the shortest distance between two different points in the data.

However, the acquirement of the *granularity* of the data is non-trivial. We should compute the distance between all the point pairs in high dimensional data space which is far beyond efficiency, and the exponential increase of the grid scale may result in losing important grid scale candidates which may yield good clustering results.

We applied a simple histogram-based approach to get reasonable grid scales for data-shrinking process. We scanned the input d-dimensional data set X once and get the set of histograms, one for each dimension:

$$H = \{h_1, h_2, ..., h_d\}.$$

Each bin of each histogram denotes the number of data points in a certain segment on this histogram.

We set up a number β as a quantity threshold. It is used in the following algorithm to help generate *Density Spans*. Here we give the definition of **density span** which will help understand our approach:

Definition 1: A **density span** is a combination of consecutive bins' segments on a certain dimension in which the amount of data points exceeds β.

For each histogram h_i, i=1,...,d, we sort its bins based on the number of data points they contain in descending order. Then we start from the first bin of the ordered bin set, merge it with its neighboring bins until the total amount of data points in these bins exceeds β. Thus a density span is generated as the combination of the segments of these bins. The operation is continued until all the non-empty bins of this histogram is in some density spans. Each histogram has a set of density spans.

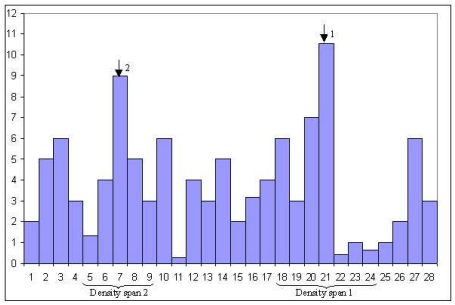


Figure 2: An example of density span acquirement

Figure 2 shows an example of this density span generation operation. Here we just demonstrate two density spans on this histogram although there are more. Bin 21 is the one with largest amount of data points. We start from Bin 21, merge it with its neighbors until the amount of data points included exceeds β. Thus density span 1 is generated. Bin 7 has the second largest amount of data points. Density span 2 is generated starting from bin 7.

We regard density spans with similar sizes as identical density spans. Once we get the set S of all the density spans from all the histograms, we sort them based on their frequencies in set S. We choose first S_n density spans as the multiple scales for the following procedure. In other words, those density spans which appear often in set S are chosen. Algorithm 1 describes the procedure of the density span generation on a certain dimension.

Algorithm 1 (Density span generation)
Input: **histogram** h_i
Output: Density span set of h_i
1) Sort the bins of h_i in the descending order;
2) Beginning from the first bin of the ordered bin set, merge it with its neighbors until the total amount of data points included exceeds β;
3) Repeat step 2 until all non-empty bins are in some Density Spans;
4) Output the density span set.

The value β depends on the size of the input data set X. Normally it can be set as a certain percentage of the amount of total data points in X. There is a balance in choosing a value for S_n: a smaller S_n can increase the precision of cluster detection, while a larger S_n can save time. The time complexity for this method is determined by the dimensionality d of X and the amount of bins B_n in each histogram. The time required to perform Algorithm 1 is $O(B_n \log B_n)$.

The proposed multiscale gridding scheme not only facilitates the determination of a proper cell size but also offers advantages for handling data sets with clusters of various densities. For example, the data set in Figure 3 has three clusters. The two clusters on the left have higher densities than the cluster on the right. The grid with a smaller cell size (shown in solid lines) can distinguish the left two clusters but fails to detect the right cluster, while the converse is true for the grid with a larger cell size (shown in dashed lines). For data sets of this kind, a multiscale gridding method is needed to distinguish all clusters.

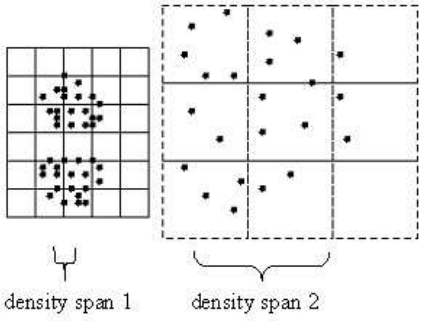


Figure 3: A data set with three clusters

4 Data Shrinking

In the *data-shrinking* step of the proposed method, each data point moves along the direction of the density gradient and the data set shrinks toward the inside of the clusters. Points are "attracted" by their neighbors and move to create denser clusters. This process is repeated until the data are stabilized or the number of iterations exceeds a threshold.

The neighboring relationship of the points in the data set is grid-based. The space is first subdivided into grid cells. Points in sparse cells are considered to be noise or outliers and will be ignored in the data-shrinking process. Assume a dense cell C with neighboring cells surrounding C. Data shrinking proceeds iteratively; in each iteration, points in the dense cells move toward the data centroid of the neighboring cells. The iterations terminate if the average movement of all points is less than a threshold or if the number of iterations exceeds a threshold.

The major motivation for ignoring sparse cells is computation time. If the grid cells are small, the number of non-empty cells can be $O(n)$, where n is the number of data points. The computation of data movement for all non-

empty cells takes a length of time quadratic to the number of non-empty cells, which is $O(n^2)$. By ignoring sparse cells in the data movement, dramatic time savings can be realized.

4.1 Space subdivision

Given the side length $\frac{1}{\kappa}$ of grid cells, the hypercube $[0,1)^d$ is subdivided into κ^d cells:

$$\{C(i_1,i_2,...,i_d) = [\frac{i_1}{\kappa},\frac{i_1+1}{\kappa}) \times [\frac{i_2}{\kappa},\frac{i_2+1}{\kappa}) \times ... \times [\frac{i_d}{\kappa},\frac{i_d+1}{\kappa}) | i_1,i_2,...,i_d \in \{0,1,...,\kappa-1\}\}. \quad (5)$$

Each cell $C(i_1,i_2,...,i_d)$ has a unique ID: $(i_1,i_2,...,i_d)$. Two *distinct* cells $C(i_1,i_2,...,i_d)$ and $C(j_1,j_2,...,j_d)$ are *neighboring cells* if $|i_k - j_k| \leq 1$ for all $k = 1,2,...,$ or d. The neighboring cells of a cell C are also called the *surrounding cells* of C. This arrangement is shown in Figure 4.

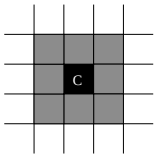


Figure 4: Surrounding cells (in gray) of the cell C (in black)

For each data point $\vec{X}_i$, the cell containing the point can be easily found; this cell is denoted as $Cell(\vec{X}_i)$. We then sort the data points into their cells to find all nonempty cells and the points contained by each. For each nonempty cell, we compute its density, defined as a fraction of the number of points in the cell over the volume of the cell. A cell is called a *sparse/dense* cell if its density is less/not less than a density threshold T_{dn1}. The selection of the density threshold T_{dn1} will be discussed in Subsection 4.4, below. Points in sparse cells are considered to be noise or outliers and will be ignored in the data-shrinking process. We then denote the set of dense cells as

$$DenseCellSet = \{C_1, C_2, ..., C_m\}. \quad (6)$$

For each dense cell C, the centroid of its points is computed:

$$DataCentroid(C) = \frac{\Sigma_{j=1}^{k} \vec{X}_{i_j}}{k}, \quad (7)$$

where $\{\vec{X}_{i_j}\}_{j=1}^{k}$ is the set of points in the cell. It is called the *data centroid* of the cell C. Each dense cell contains its own points and data centroid. The computational process involved in finding the dense cells, their points, and their centroids takes time $O(n \log n)$. The space occupied by the dense cells is $O(n)$. The process of space subdivision is repeated at the beginning of each data-movement iteration.

In high-dimensional spaces, ignoring the sparse cells can be problematical. Figure 5(a) illustrates four points in a two-dimensional grid. The four points are clustered near a vertex of the grid and are separated in four neighboring cells. In the d-dimensional Euclidean space $\mathbb{R}^d$, there may be 2^d points distributed in a similar manner. These points should have an influence on the data-shrinking process but will be ignored because they are separated in different cells. To address this issue, we choose two interleaved grids for a given cell size. An example of such interleaved grids is given in Figure 5(a) and (b). The data-shrinking process is conducted alternately on the two grids.

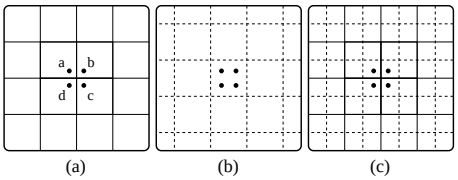


Figure 5: (a) Points a, b, c, and d are located near a vertex of the grid and are separated in four neighboring cells; (b) another grid with the same cell size; (c) the two grids (shown respectively by solid and dashed lines) in the same plane

4.2 Data movement in a single iteration

Data movement is an iterative process intended to move data gradually toward a goal of increased cluster density. This data-movement process is conducted alternately on two interleaved grids in alternating iterations.

In each iteration, points are "attracted" by their neighbors and move toward the inside of the clusters. Each point in a cell C has neighboring points in C or in the cells surrounding C. The movement of a point can be intuitively understood as analogous to the attraction of a mass point by its neighbors, as described by Newton's Law of Universal Gravitation. Thus, the point moves toward the centroid of its neighbors. However, data movement thus defined can cause an evenly-distributed data set to be condensed in a piece-wise manner. For example, for each point in Figure 6(a), the centroid of its neighbors is in the center of a grid cell, causing all points to be attracted to the centers of the grid cells. After the data-movement procedure, the data set in Figure 6(a) becomes the isolated points shown in Figure 6(b).

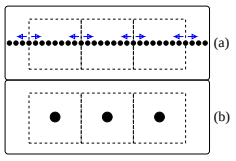


Figure 6: A part of a data set (a) before movement and (b) after movement. The arrows in (a) indicate the direction of motion of the points. The grid cells are shown in dashed lines.

Our solution to the above problem is to treat the points in each cell as a rigid body which is pulled as a unit toward the data centroid of those surrounding cells which have more points. Therefore, all points in a single cell participate in the same movement. This approach not only solves the problem of piece-wise condensing but also saves time.

Formally, suppose the data set at the beginning of ith iteration becomes

$$\{\vec{X}_1^i, \vec{X}_2^i, ..., \vec{X}_n^i\},$$

and the set of dense cells is

$$DenseCellSet^i = \{C_1^i, C_2^i, ..., C_m^i\}. \quad (9)$$

Respectively, we assume that the dense cells have

$$n_1, n_2, ..., n_m,$$

points and their data centroids are

$$\vec{\Phi}_1, \vec{\Phi}_2, ..., \vec{\Phi}_m. \quad (11)$$

For a dense cell C_j^i, we suppose that its surrounding dense cells are $C_{j_k}^i$ for $k = 1, 2, ..., w$. Then the data centroid of these surrounding cells is

$$\frac{\Sigma_{k=1}^w n_{j_k} \times \vec{\Phi}_{j_k}}{\Sigma_{k=1}^w n_{j_k}}, \quad (12)$$

which is denoted as $\vec{\Phi}_j^s$. The movement for cell C_j^i in the ith iteration is

$$Movement(C_j^i) = \begin{cases} \vec{\Phi}_j^s - \vec{\Phi}_j \ \ if \| \vec{\Phi}_j^s - \vec{\Phi}_j \| \geq T_{mv} \times \frac{1}{\kappa} \\ \qquad\qquad and \quad \Sigma_{k=1}^w n_{j_k} > n_j; \\ 0 \qquad\qquad otherwise, \end{cases} \quad (13)$$

where $\| \vec{\Phi}_j^s - \vec{\Phi}_j \|$ is the distance between the two centroids $\vec{\Phi}_j^s$ and $\vec{\Phi}_j$, T_{mv} is a threshold to ensure that the movement is not too small, and $\frac{1}{\kappa}$ is the side length of grid cells, as discussed in last subsection. T_{mv} is usually a number between 0.5 and $0.5 \times \sqrt{d}$. Formula 13 states that, if the distance between $\vec{\Phi}_j^s$ and $\vec{\Phi}_j$ is not too small and the surrounding cells have more points, then cell C_j^i will be translated such that its data centroid is moved to the data centroid of the surrounding dense cells; otherwise, cell C_j^i remains static. The movement for each cell indicates the approximate direction of the density gradient around the cell. After movement, a data point $\vec{X}$ in cell C_j^i is moved to $\vec{X} + \vec{\Phi}_j^s - \vec{\Phi}_j$.

To compute the movement for a dense cell C_j^i, we browse the set of dense cells to find its surrounding cells and then calculate the movement. The computation takes $O(m)$ time. It then takes $O(n_j)$ time to update the points in the cell. Therefore in the ith iteration, the time used to move all points is $O(m^2 + n)$. Thus, the time required for the ith iteration is $O(m^2 + n \log n)$, where $O(n \log n)$ time is used for the subdivision of space.

Two examples are given in Figure 7. Geometrically, the data movement which occurs in a single iteration has two effects. For a data set covering a manifold [1] with a boundary in the d-dimensional Euclidean space $\mathbb{R}^d$, data movement squeezes the manifold from its boundary (see Figure 7(a)). For the interior of a manifold covered by a data set, data movement smoothes out the corners (see Figure 7(b)).

[1] A manifold is a topological space which is locally Euclidean.

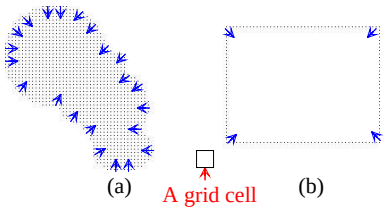


Figure 7: Movement of two data sets which cover (a) a region and (b) a closed line, respectively.

4.3 Termination of shrinking

Ideally, for a data set covering a manifold with a boundary, the shrinking process pushes the boundary points inward until the manifold is reduced to its skeleton. If the skeleton is also a manifold with a boundary, it is skeletonized again. This process is repeated until a manifold with no boundary is produced, as shown in Figure 8. However, most data sets from real-world applications do not have well-defined shapes in high-dimensional spaces. The data sets resulting from the shrinking process may also not have well-defined shapes. In general, the shrinking process produces individual clusters which are condensed and therefore widely separated, facilitating cluster detection.

Figure 8: Repeated skeletonization

The average movement of all points in each iteration is checked to determine the stability of the data set. Suppose that in the ith iteration, the movements for the n points are $\vec{v}_1, \vec{v}_2, ..., \vec{v}_n$, respectively. Then the average movement is $\frac{\Sigma_{j=1}^n \|\vec{v}_j\|}{n}$. If the average movements for two consecutive iterations are both less than $T_{amv} \times \frac{1}{\kappa}$, where T_{amv} is a threshold, then the data set is considered *stabilized* and the shrinking process is terminated.

4.4 Time and space analysis

Throughout the shrinking process, we need to keep track of the locations of all points, which collectively occupy $O(n)$ space. Data points are assigned to grid cells. Each dense grid cell serves as a container for its points, together with their centroid and movement. The data structure represented by the dense cells occupies $O(n)$ space; therefore, the total space needed is $O(n)$. The time required for each iteration is $O(m^2 + n \log n)$, where m is the number of dense cells. Since the maximum number of iterations is T_{it}, the total running time is $O(T_{it}(M^2 + n \log n))$, where M is the maximum number of dense cells in all iterations.

The value M, representing the maximum number of dense cells, has a significant impact on running time. M can be controlled through the selection of a density threshold T_{dn1}. The number of data points in a dense cell must be no less than the product of T_{dn1} and the volume of the cell, or $T_{dn1} \times (\frac{1}{\kappa})^d$. Thus the number of dense cells must not exceed $\frac{n}{T_{dn1} \times (\frac{1}{\kappa})^d}$. Given a desired value $\tilde{M}$, we can choose a value T_{dn1} such that $\frac{n}{T_{dn1} \times (\frac{1}{\kappa})^d} \leq \tilde{M}$, thus en-

suring that the number of dense cells will not exceed $\tilde{M}$. However, if the densities of most cells happen to fall below a threshold T_{dn1} chosen via this method, the data-shrinking process will be unproductive. Alternatively all non-empty cells can be sorted by the number of data points they contain. The density threshold T_{dn1} is then chosen so that the first $\tilde{M}$ cells are dense cells. Cases may occur where, for a given grid-cell side length $\frac{1}{\kappa}$, most non-empty cells will be very sparse, containing only one or two points each. In such instances, the side length $\frac{1}{\kappa}$ is too small and a larger-scale grid should be used.

5 Cluster detection

Since the data-shrinking process generates individual clusters which are condensed and widely separated, it can be used as a preprocessing with any cluster-detection algorithm. In this paper, we use a simple grid-based cluster-detection method to test the data-shrinking process.

For a given cell-side length $\frac{1}{\kappa}$, after the data-shrinking process is performed on the input data set, we find the dense cells. Neighboring dense cells are connected and a neighboring graph of the dense cells is constructed. Each connected component of the neighboring graph is a cluster.

The cluster-detection method is conducted on two interleaved grids. This avoids the problem caused by points clustered near a vertex of a grid and separated in different cells, as discussed in Subsection 4.1. Let T_{dn2} be a density threshold. A cell in either of the two interleaved grids is called a *dense cell* if its density is no less than T_{dn2}. Let DC_1 and DC_2 be the dense cell sets of the two interleaved grids, respectively. Two cells $C_1 \in DC_1$ and $C_2 \in DC_2$ are called *neighbors* if $C_1 \bigcap C_2 \neq \emptyset$. The neighboring graph of dense cells, G, is a pair $G =< DC_1 \bigcup DC_2, E >$, where E is the set of neighboring pairs in $DC_1 \bigcup DC_2$. The edge set E can be represented by a matrix. Let $|DC_1| = m_1$, $|DC_2| = m_2$, and $DC_1 \bigcup DC_2 = \{C_i\}_{i=1}^{m_1+m_2}$. Then $E = (E_{ij})_{(m_1+m_2)\times(m_1+m_2)}$, where

$$E_{ij} = \begin{cases} 1 & \text{if } C_i \text{ and } C_j \text{ are neighbors;} \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

We then run a breadth-first search algorithm (see pages 469-472 in [7]) to find the components of graph G.

The time and space required for the breadth-first search algorithm are both $O(|DC_1 \bigcup DC_2| + |E|)$. To construct graph G, the time and space needed are $O((m_1 + m_2)^2)$. The total time and space required for the cluster-detection algorithm are therefore $O((m_1 + m_2)^2)$. Since the data-shrinking process is performed first, the number of dense cells, defined as $m_1 + m_2$, is greatly reduced, which makes our cluster-detection algorithm particularly useful for real data sets of large size.

6 Cluster evaluation and selection

Most conventional clustering validity measurements [6, 17, 18, 19] evaluate clustering algorithms by measuring the overall quality of the clusters. However, each clustering algorithm has its advantages and disadvantages. For a data set with clusters of various sizes, densities, or shapes, different clustering algorithms are best suited to detecting the clusters of different types in the data set. No single approach combines the advantages of the various clustering algorithms while avoiding their disadvantages. In this section, we introduce a cluster-wise measurement which provides an evaluation method for individual clusters.

A cluster in a data set is a subset in which the included points have a closer relationship to each other than to points outside the cluster. In the literature [6, 17], the intra-cluster relationship is measured by *compactness* and the inter-cluster relationship is measured by *separation*. Compactness is a relative term; an object is compact in comparison to a looser surrounding environment. We use the term *compactness* to measure the quality of a cluster on the basis of intra-cluster and inter-cluster relationships. This definition of compactness is used to evaluate clusters detected at different scales and to then select the best clusters as the final result.

6.1 Compactness graphing

We first define compactness using a weighted graph. In this subsection, let $G =< V, E >$ be a fixed graph, where V is the vertex set and E is the edge set. Let $w : E \to \mathbb{R}^+$ be a weight function on the edge set. We use the terms *internal connecting distance (ICD)* and *external connecting distance (ECD)* to measure the closeness of the internal and external relationships, respectively. Compactness is then defined as the ratio of the external connecting distance over the internal connecting distance.

Definition 2: For a connected subset S of V, let $MST(S)$ be a minimum spanning tree of the minimal subgraph containing S. The **internal connecting distance (ICD)** of S, denoted as $ICD(S; G, w)$, is defined as the length of a longest edge of $MST(S)$.[2] The **external connecting distance (ECD)** of S, denoted as $ECD(S; G, w)$, is defined as the length of a shortest edge connecting S and $V - S$. The **compactness** of S, denoted as $Compactness(S; G, w)$, is defined as

$$Compactness(S; G, w) = \frac{ECD(S; G, w)}{ICD(S; G, w)}. \quad (15)$$

S is called a **compact vertex set** if its compactness is greater than one.

The definition of the external connecting distance is quite straightforward. The internal connecting distance can be interpreted as the shortest distance which maintains a connected set. For a connected subset S of V, if we remove all edges longer than $ICD(S; G, w)$ from G, then S remains connected because none of the edges of a minimum spanning tree $MST(S)$ containing S is longer than $ICD(S; G, w)$; if we remove all edges not shorter than

[2] It can be proved that, for two minimum spanning trees of a given graph, their longest edges are equal in length.

$ICD(S; G, w)$ from G, then S will be disconnected. To evaluate a data set in a low-dimensional Euclidean space, we first construct its Delaunay graph [3]. Compactness is then defined on the Delaunay graph. There is no efficient way to construct Delaunay graphs for data sets in high-dimensional spaces. However, we can define compactness on the complete graphs of these data sets if they are of moderate size.

6.2 Grid-based compactness

The definition of compactness offered above suffers from two drawbacks. First, it is sensitive to noise. For example, the compactness of the two clusters in Figure 9 is lowered by the scatter of the noisy points. Second, as noted above, Delaunay graphs can not be efficiently constructed for high-dimensional spaces. In these instances, compactness must be defined on complete graphs, a process which requires quadratic space and time. These two problems can be easily remedied with a grid-based approach. Given an input data set and a defined scale, we first find the dense cells of two interleaved grids at this scale. Compactness is then defined on the complete graph of the dense cells. Because the sparse cells are ignored, running time is reduced and the result is not noise-sensitive.

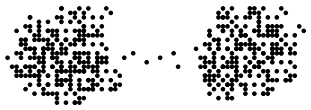


Figure 9: Two clusters with noisy points in between

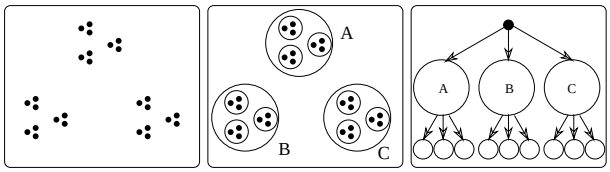
A more detailed description of the determination of compactness is as follows. We first condense the input data by running the shrinking process with a selected cell size. Then, as we discussed in Section 5, clusters are detected as the connected components of the neighboring graph of the dense cells. Let DC be the set of dense cells produced by the shrinking process. We define the complete graph of DC as $DG =< DC, E >$, where E is the set of pairs of cells in DC. The length of each edge in E is defined as the Euclidean distance between the data centroids of the two cells connected by the edge. The compactness of each detected cluster is defined on the complete graph DG. Compactness as defined in this process is the compactness of the clusters of the data set after shrinking, termed *compactness after shrinking*. However, this measure of compactness may not truly represent the quality of the clusters in the original data. As an alternative, we can map the clusters to their original locations in the space and then measure their compactness, giving a measure of *compactness before shrinking*.

To compute the compactness of each cluster, we first compute its internal and external connecting distances. To compute the internal connecting distance of a specific cluster with v cells, first construct the minimum spanning tree of the cluster using Prim's algorithm (see pages 505-510 in [7]). We then browse the edge set of the minimum spanning tree to find the internal connecting distance. The computation takes $O(v^2)$ time. To compute the external connecting distance, we compute the shortest distance connecting cells in the cluster with cells outside the cluster. The computation takes $O(v \times (|DC| - v))$. The time required to compute the compactness of the cluster is $O(v \times |DC|)$, and the total time to compute the compactness of all clusters is thus $O(|DC|^2)$.

6.3 Evaluation and selection of multiscale clusters

In evaluating a given data set, we run the data-shrinking and cluster-detection processes using a sequence of grids of selected cell sizes as mentioned in Section 3. We compute the compactness-before-shrinking of the clusters detected at all scales. Those clusters with compactness exceeding a specified threshold will be output as the final result.

Within the final result, a cluster can be a subset of another cluster. For example, the clusters of the data set in Figure 10(a) form a tree in Figure 10(c). For all clusters to form a tree, one of the following must be true for any two clusters C_1 and C_2: $C_1 \subseteq C_2$, $C_2 \subseteq C_1$, or $C_1 \bigcap C_2 = \emptyset$. Furthermore, for any graph $G =< V, E >$ with a weight function w on the edge set, if two subsets S_1 and S_2 of V have compactnesses greater than one, we can prove that $S_1 \subseteq S_2$, $S_2 \subseteq S_1$, or $S_1 \bigcap S_2 = \emptyset$. In situations where an inclusive relationship exists, all compact vertex sets form a tree.



(a) A data set (b) Clusters (c) Cluster tree

Figure 10: The cluster tree of a data set

Our definition of compactness can be used to unify multiple clustering algorithms, exploiting their advantages and avoiding their disadvantages. Multiple clustering algorithms can be run on a dataset, the compactness of the detected clusters compared, and the best clusters output.

7 Experiments

Comprehensive experiments were conducted to assess the accuracy and efficiency of the proposed approach. Our experiments were run on SUN ULTRA 60 workstations with the Solaris 5.8 system. To demonstrate the functioning of the shrinking process, we will first discuss experiments conducted using a 2D data set. Trials using data sets from real-world applications comparing to other algorithms such as CURE and OPTICS are offered as a demonstration of the accuracy of the proposed approach. Finally, experiments are conducted to demonstrate how the shrinking preprocessing solely will improve the performance of well known algorithms such as OPTICS, CURE and BIRCH.

In our experiments, T_{mv} is set at $0.5 \times \sqrt{d}$, where d is the number of dimensions. T_{dn1} is defined dynamically as one-third of the average density of the nonempty cells

in each iteration; T_{dn2} is defined similarly. Other parameters are optimized as follows. For each parameter, we select several candidates, run the algorithm on the candidate parameters, and compare the compactness of the detected clusters. The best candidates are then selected as the values for these parameters.

7.1 Experiments on 2D datasets

We first conducted experiments on 2-dimensional data sets as intuitive demonstrations for data shrinking preprocessing procedure. Due to the space limitation, here we just present the shrinking result on one data set DS_1 which has 2682 points including noisy data. There are two clusters in the data with one is half-embraced by the other. The shrinking process generates two well-separated clusters of arbitrary shape and filters outliers, thus facilitating cluster detection.

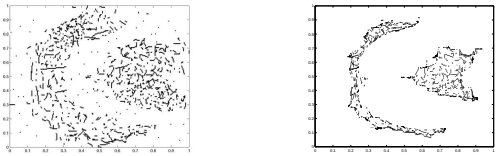


Figure 11: Shrinking process on the data set DS_1 with cell size 0.1×0.1. (a) 2-dimensional data set DS_1, (b) the data set after the shrinking process

7.2 Experiments on real data

Finally, we demonstrate that our algorithm has been found to yield encouraging results in real-world clustering problems. We tested our approach on three data sets from real applications and demonstrate its accuracy for clustering compared to CURE and OPTICS. The accuracy of a detected cluster was measured according to *precision* and *recall*. For a detected cluster C_i^s and a real cluster C_i^o, we define the precision of C_i^s with respect to C_i^o as $\frac{|C_i^s \bigcap C_i^o|}{|C_i^s|}$ and the recall as $\frac{|C_i^s \bigcap C_i^o|}{|C_i^o|}$. C_i^s is called a corresponding cluster of C_i^o if the precision and recall of C_i^s with respect to C_i^o are high.

7.2.1 Algorithms

CURE: We used the implementation of CURE provided to us by Michael Steinbach from University of Minnesota. It requires three input parameter options: -k option is for the number of clusters, -α is for alpha parameter of CURE, and -r is the number of representative points of the cluster. To compare CURE with our algorithm fairly, we applied different values of those parameters extensively and adopted the best clustering results. Since we used CURE mainly to compare the accuracy of its clustering result with ours, we didn't take consideration of the partition number parameter p for speedup mentioned in [10].

OPTICS: We adopted the implementation of OPTICS provided by Peer Kroeger. OPTICS does not produce a clustering of a data set explicitly. It instead creates an augmented ordering of the data set representing its density-based clustering structure. We can roughly estimate the generated clusters by the observation of its results. Since OPTICS claims that the reachability-plot is rather insensitive to the input parameters (the generating distance eps and the value for MinPts)[4], we set the parameter values for OPTICS just to be "large" enough to yield a good result.

BIRCH: We also used the implementation of BIRCH[27] to show how shrinking preprocessing will affect the performance of BIRCH. The implementation performs preclustering and then uses a centroid-based hierarchical clustering algorithm. The parameter values are set to the default values suggested in [27].

Our algorithm: Our clustering version is based on the algorithm described in previous sections which includes *Data Shrinking*, *Cluster Detection* and *Cluster Evaluation* and *Selection*. First the testing data sets are shrunk so that natural clusters become more condensed, resulting in potentially much easier and more efficient cluster detection. Then clusters are detected at different scales. A cluster-wise evaluation measurement is applied to compare clusters at those scales and the final result is acquired.

7.2.2 Data sets and clustering results

The three data sets were obtained from UCI Machine Learning Repository [5]. The first data set, Wine Recognition data, contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. It contains 178 instances, each of which has 13 features, including alcohol, magnesium, color intensity, etc. The data set has three clusters, labelled as C_i^o, with $i = 1, 2, 3$. Our algorithm detected three corresponding clusters, labelled as C_i^s, with $i = 1, 2, 3$. Table 1 shows the clustering results of our algorithm.

	i=1	i=2	i=3
$\lvert C_i^o \rvert$	59	71	48
$\lvert C_i^s \rvert$	53	52	46
$\lvert C_i^s \bigcap C_i^o \rvert$	53	51	43
precision(%)	100	98.08	93.48
recall(%)	89.83	71.83	89.58

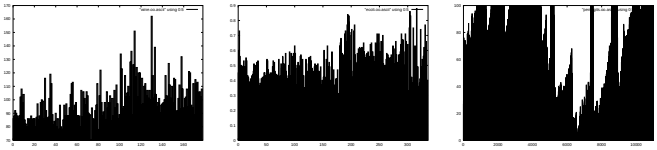
Table 1: Clustering results of our algorithm for Wine data

We applied CURE algorithm on the Wine Recognition data set, setting parameter values to different values extensively. We set the cluster number parameter k to 3 based on the ground truth of the Wine Recognition data set, set the shrinking factor α to the set of [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], and set the number of representative points r to the set of [2, 4, 5, 10, 20, 30, 40, 50, 60]. We found that the clustering result is best when the (α, r) pair is (0.3, 30), (1, 20), (0.9, 10) or (0.5, 40). Because of space limitation, here we just present one of the best results of CURE. Table 2 shows the clustering results of CURE algorithm when α is equal to 0.3, and r is 30.

Figure 12(a) shows the cluster-ordering of OPTICS for the Wine data. From the figure we can see there are roughly

	i=1	i=2	i=3
$\|C_i^o\|$	59	71	48
$\|C_i^s\|$	72	50	46
$\|C_i^s \bigcap C_i^o\|$	54	41	26
precision(%)	75.00	82.00	56.52
recall(%)	91.52	57.77	54.16

Table 2: Clustering result of CURE for Wine data as α=0.3 and r=30



(a) (b) (c)

Figure 12: Testing result of OPTICS for (a) Wine data with eps=200 and MinPts=10, for (b) Ecoli data with eps=100 and MinPts=10 and for (c) Pendigits data with eps=1000 and MinPts=100.

9 clusters generated.

We can see that our algorithm's accuracy for clustering is better than that of CURE based on the comparison between table 1 and 2.

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
$\|C_i^o\|$	143	77	52	35	20	5	2	2
$\|C_i^s\|$	135	22	68	49	11	N/A	N/A	N/A
$\|C_i^s \bigcap C_i^o\|$	130	22	43	32	10	N/A	N/A	N/A
precision(%)	96.30	100	63.24	65.31	90.91	N/A	N/A	N/A
recall(%)	90.91	28.57	82.69	91.43	50.00	N/A	N/A	N/A

Table 3: Clustering result of our algorithm for Ecoli data

The second data set, Ecoli, contains data regarding Protein Localization Sites. This set is made up of 336 instances, with each instance having seven features. Table 3 presents the clustering results. The real clusters C_6^o, C_7^o, and C_8^o do not have corresponding clusters detected by our algorithm. These clusters have few points, located in sparse cells, and thus are ignored and discarded in the shrinking and cluster-detection processes of our algorithm.

We applied CURE algorithm on the Ecoli data set, setting parameter values to different values extensively. According to the ground truth of the Ecoli data set, there are 8 clusters in it. However, three of the clusters are too small which have only 2, 3 and 5 data points in them respectively. So we set the cluster number parameter k to 5(we also set k to 8 and found that the clustering result is not as good as those with k as 5), set the shrinking factor α to the set of [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], and set the number of representative points r to the set of [2, 4, 6, 8, 10, 15, 20, 30, 40, 50, 60, 70]. We found that the clustering result is best when the (α, r) pair is (0.2, 30), (0.9, 15) or (0.8, 20). Because of space limitation, here we just present one of the best results of CURE. Table 4 shows the clustering results of CURE algorithm when α is equal to 0.2, and r is 30.

Our algorithm's accuracy for clustering on Ecoli data is also better than that of CURE based on the comparison between table 3 and 4.

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
$\|C_i^o\|$	143	77	52	35	20	5	2	2
$\|C_i^s\|$	120	67	32	N/A	N/A	3	N/A	N/A
$\|C_i^s \bigcap C_i^o\|$	115	41	30	N/A	N/A	3	N/A	N/A
precision(%)	95.83	61.19	93.75	N/A	N/A	100	N/A	N/A
recall(%)	80.41	53.24	57.69	N/A	N/A	60.00	N/A	N/A

Table 4: Clustering result of CURE for Ecoli data as α=0.2 and r=30

Figure 12(b) shows the cluster-ordering of OPTICS for the Ecoli data. From the figure we can see there are roughly 12 clusters generated.

The third data set is Pendigits, or Pen-Based Recognition of Handwritten Digits. It was created by collecting 250 samples from 44 writers. It has two subsets used, respectively, for training and testing. For the purpose of this experiment, we have combined these two subsets, resulting in a combined dataset with 10992 instances, each containing 16 attributes. The data set has ten clusters, C_i^o for $i = 1, 2, ..., 10$. Our algorithm detected eight clusters C_i^s for $i = 1, 2, ..., 8$. The first six detected clusters, C_1^s through C_6^s, correspond to C_1^o through C_6^o respectively. The seventh detected cluster, C_7^s, corresponds to C_7^o and C_8^o; and the last detected cluster, C_8^s, corresponds to C_9^o and C_{10}^o. Table 5 shows the clustering results for this data set. These results demonstrate that our approach can effectively detect clusters in data sets from real applications.

We applied CURE algorithm on the Pendigits data set, setting parameter values to different values extensively. We set the cluster number parameter k to 10, set the shrinking factor α to the set of [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1], and set the number of representative points r to the set of [2, 5, 8, 10, 20, 30, 50, 70, 100, 200, 300, 500]. We found that the clustering results are best when the (α, r) is set to (0.4, 50), (0.9, 5) or (0.9 8). Because of space limitation, here we just present one of the best results of CURE. Table 6 shows the best clustering results of CURE algorithm.

Again our algorithm's accuracy for clustering on Pendigits data is better than that of CURE based on the comparison between table 5 and 6.

Figure 12(c) shows the cluster-ordering of OPTICS for the Pendigits data. From the figure we can see there are roughly 8 clusters generated which is similar to the clustering result of our algorithm. However, the sizes of the clusters do not match the ground truth very well.

7.3 Experiments on how shrinking preprocessing improves clustering algorithms

Finally, we will demonstrate how the shrinking preprocessing will solely improve the performance of well known clustering algorithms.

OPTICS: First we will show the difference between the testing results of OPTICS on wine data and pendigits data.

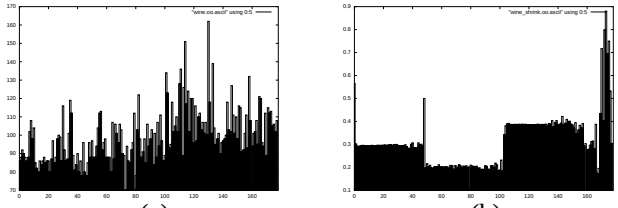
From Figure 13 we can see that after shrinking preprocessing, the cluster-ordering is much more significant than that without shrinking preprocessing. And the curve shown

	i=1 j=1	i=2 j=2	i=3 j=3	i=4 j=4	i=5 j=5	i=6 j=6	i = 7 j = 7	i = 8 j = 7	i = 9 j = 8	i = 10 j = 8
$\|C_i^o\|$	1143	1144	1055	1056	1055	1055	1143	1055	1144	1142
$\|C_j^s\|$	1098	1179	629	1051	480	833	2379	2379	2376	2376
$\|C_j^s \bigcap C_i^o\|$	1094	1084	625	1046	480	606	709	1049	1132	912
precision(%)	99.63	91.94	99.36	99.52	100	72.75	29.80	44.09	47.64	38.38
recall(%)	95.71	94.76	59.24	99.05	45.50	57.44	62.03	99.43	98.95	79.86

Table 5: Clustering result of our algorithm for Pendigits data. For the last four columns, the corresponding relationship is two-to-one: two real clusters correspond to one detected cluster.

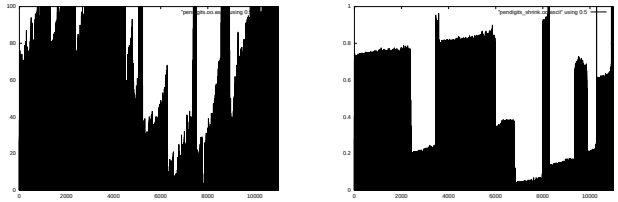
	i=1 j=1	i=2 j=2	i=3 j=3	i=4 j=4	i=5 j=5	i=6 j=6	i = 7 j = 7	i = 8 j = 7	i = 9 j = 7	i = 10 j = 7
$\|C_i^o\|$	1143	1144	1055	1056	1055	1055	1143	1055	1144	1142
$\|C_j^s\|$	897	758	462	835	125	28	4121	4121	4121	4121
$\|C_j^s \bigcap C_i^o\|$	897	715	461	835	125	28	816	885	954	780
precision(%)	100	94.32	99.78	100	100	100	19.80	21.47	23.14	18.92
recall(%)	78.47	62.50	43.69	79.07	11.84	2.65	71.39	83.88	83.39	68.30

Table 6: Clustering result of CURE for Pendigits data as α=0.4 and r=50 . For the last four columns, the corresponding relationship is four-to-one: four real clusters correspond to one detected cluster.



(a) (b)

Figure 13: Testing result of OPTICS for Wine data (a) without shrinking preprocessing (b) after shrinking preprocessing



(a) (b)

Figure 14: Testing result of OPTICS for Pendigits data (a) without shrinking preprocessing (b) after shrinking preprocessing

on Figure 13 (b) matches the ground truth (3 clusters of 59, 71, 48 data points) much better than the original curve on Figure 13 (a). The reason is that after shrinking preprocessing, the clusters are more condensed and outliers are further isolated which make the clustering algorithms more efficient and more effective.

Figure 14 shows the curve difference of cluster-ordering without shrinking preprocessing and after shrinking preprocessing. Again after shrinking preprocessing, the cluster-ordering is much more significant than that without shrinking preprocessing.

CURE: We tested the CURE algorithm on several data sets after shrinking preprocessing to see its effect. Table 7 shows the clustering results of CURE algorithm on Wine data after shrinking preprocessing when α is equal to 0.3, and r is 30. Comparing Table 7 to the original clustering result Table 2, we can see that the recalls of the clusters generated from CURE on the Wine data after shrinking preprocessing are comparable to those generated from CURE on the original Wine data, while the precisions of the clusters are much better than the original ones.

Table 8 shows the clustering results of CURE algorithm on Ecoli data after shrinking preprocessing when α is equal to 0.2, and r is 30. The qualities of the clusters generated from CURE on the Ecoli data after shrinking preprocessing are better than those of the original clusters(see Table 4).

	i=1	i=2	i=3
$\|C_i^o\|$	59	71	48
$\|C_i^s\|$	48	43	29
$\|C_i^s \bigcap C_i^o\|$	48	43	28
precision(%)	100	100	96.55
recall(%)	81.25	60.56	58.33

Table 7: Clustering result of CURE for Wine data after shrinking preprocessing as α=0.3 and r=30

BIRCH: We also used the implementation of BIRCH provided to us by the authors of [27] to show how shrinking preprocessing will affect the performance of BIRCH on different data. Due to the space limitation, here we just show the testing result on ecoli data mentioned in previous sections. The ground truth is that the ecoli data contains 8 natural clusters, with the sizes of 143, 77, 52, 35, 20, 5, 2, 2. First we applied the BIRCH algorithm directly on the data, resulting in 8 clusters with the sizes of 133, 93, 74, 24, 6, 3, 2, 1. Then we applied BIRCH again on the data with shrinking preprocessing, and get 8 clusters with the sizes of 145, 100, 70, 9, 6, 3, 2,1. From the comparison of the two different clustering results, we can see that the major clusters generated from the shrinking preprocessing involved version match the ground truth better than those generated from the original BIRCH algorithm.

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
$\|C_i^o\|$	143	77	52	35	20	5	2	2
$\|C_i^s\|$	109	72	36	N/A	4	N/A	N/A	N/A
$\|C_i^s \bigcap C_i^o\|$	105	44	34	N/A	4	N/A	N/A	N/A
precision(%)	96.33	61.11	94.44	N/A	100	N/A	N/A	N/A
recall(%)	73.42	57.14	65.38	N/A	20.00	N/A	N/A	N/A

Table 8: Clustering result of CURE for Ecoli data after shrinking preprocessing as α=0.2 and r=30

8 Conclusion and discussion

In this paper, we first presented a novel data preprocessing technique called $shrinking$ which optimizes the inner structure of data inspired by the Newton's Universal Law of Gravitation. Then, we applied it and proposed a novel data analysis method which consists of three steps: data shrinking, cluster detection, and cluster evaluation and selection. The method can effectively and efficiently detect clusters of various densities or shapes in a noisy data set of any dimensions.

The data-shrinking process still poses many open issues. As discussed above, the shrinking process as applied to a data set of well-formed shape is a repeated skeletonizing process which transforms the data set into a shape with no boundary. However, most real-world, high-dimensional data sets do not have well-defined shapes. It is therefore of both theoretical and practical interest to fully understand how the shape of a real data set is transformed during the shrinking process. This understanding would provide insights into the geometrical and topological properties of high-dimensional data sets. An analytical method is also needed to estimate the number of iterations necessary for real data to reach stability during the shrinking process. Such a method could open the way to a faster shrinking process.

References

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and J. Park. Fast algorithms for projected clustering. In *Proceedings of the ACM SIGMOD CONFERENCE on Management of Data*, pages 61–72, Philadelphia, PA, 1999.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 94–105, Seattle, WA, 1998.

[3] N. Ahuja. Dot pattern processing using voronoi neighborhoods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(3):336–343, May 1982.

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

[5] S. D. Bay. *The UCI KDD Archive [http://kdd.ics.uci.edu]*. University of California, Irvine, Department of Information and Computer Science.

[6] Chi-Farn Chen, Jyh-Ming Lee . The Validity Measurement of Fuzzy C-Means Classifi er for Remotely Sensed Images. In *Proc. ACRS 2001 - 22nd Asian Conference on Remote Sensing*, 2001.

[7] T. H. Cormen, C. E. Leiserson, , and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[8] D. Barbara, W. DuMouchel, C. Faloutsos, P. J. Haas, J. H. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Servcik. *The New Jersey data resuction report*. Bulletin of the Technical Committee on Data Engineering, 1997.

[9] M. Ester, K. H.-P., J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[10] S. Guha, R. Rastogi, and K. Shim. Cure: An effi cient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD conference on Management of Data*, pages 73–84, Seattle, WA, 1998.

[11] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *Proceedings of the IEEE Conference on Data Engineering*, 1999.

[12] A. Hinneburg and D. A.Keim. An effi cient approach to clustering in large multimedia databases with noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 58–65, New York, August 1998.

[13] J. MacQueen. Some methods for classifi cation and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I,Statistics.*, 1967.

[14] A. Jain, M. Murty, and P. Flyn. Data clustering: A review. *ACM Computing Surveys*, 31(3), 1999.

[15] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[16] L.J. Cowen, C.E. Priebe. Randomized non-linear projections uncover high-dimensional structure. pages 319–331, 1997.

[17] Maria Halkidi, Michalis Vazirgiannis. A Data Set Oriented Approach for Clustering Algorithm Selection. In *PKDD*, 2001.

[18] Maria Halkidi, Michalis Vazirgiannis. Clustering Validity Assessment: Finding the Optimal Partitioning of a Data Set. In *ICDM*, 2001.

[19] Maria Halkidi,Yannis Batistakis, Michalis Vazirgiannis. Clustering Algorithms and Validity Measures. In *SSDBM*, 2001.

[20] R. T. Ng and J. Han. Effi cient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th VLDB Conference*, pages 144–155, Santiago, Chile, 1994.

[21] T. Redman. *Data Quality: Management and Technology.* Bantam Books, 1992.

[22] Rothman, Milton A. *The laws of physics*. New York, Basic Books, 1963.

[23] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the ACM SIGMOD conference on Management of Data*, pages 154–164, Seattle, WA, 1998.

[24] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of the 24th International Conference on Very Large Data Bases*, 1998.

[25] W. Wang, J. Yang, and R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the 23rd VLDB Conference*, pages 186–195, Athens, Greece, 1997.

[26] Warren Viessman Jr., Gary L. Lewis. *Introduction to Hydrology, 4/e*. Prentice Hall, 1996.

[27] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Effi cient Data Clustering Method for Very Large Databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, 1996.