DYNAMIC DATA MINING ON
MULTI-DIMENSIONAL DATA

By

Yong Shi

August 2005

A DISSERTATION PROPOSAL SUBMITTED TO THE
FACULTY OF THE GRADUATE SCHOOL OF STATE
UNIVERSITY OF NEW YORK AT BUFFALO
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY
Acknowledgments

The writing of a dissertation is a lonely and isolating experience, yet it is obviously not possible without the support of numerous people. Thus my sincere gratitude goes to my family, my advisor, and all my friends, for their support and patience over the last few years.

My greatest acknowledgement is to my advisor, Dr. Aidong Zhang, for her guidance, support, encouragement and patience; without her guidance, the completion of my dissertation would just not have been possible. I would like to express my deepest appreciation for support to Dr. Yuqing Song. Without his help, so generously given, I would never be productive as I am. I am also very grateful to my other committee members, Dr. Xin He and Dr. Jinhui Xu, for numerous discussions and inspirational comments.

In addition, I wish to acknowledge Dr. Ling Bian of Geography Department of SUNY at Buffalo who generously took the time to review my dissertation and made suggestions for
improvement. Her suggestions proved grate valuable in improving the final version of the dissertation.

I appreciate all the contributions of my colleagues. I wish to express my gratitude to my friends who made me enjoy staying at Buffalo. Amongst many others I would like to thank especially the following people for their support and useful comments: Chun Tang, Lei Zhu, Wei Wang, Zhaofan Ding, Pengjun Pei, Daxin Jiang, and Xian Xu.

Last, but by no means least I would like to thank my family for their love and nurture in all my years.

The work in this dissertation was carried out at the Department of Computer Science and Engineering. I am grateful to National Science Foundation (NSF) and National Institute of Health (NIH) for their financial support.
Abstract

The generation of multi-dimensional data has proceeded at an explosive rate in many disciplines with the advance of modern technology, which greatly increases the challenges of comprehending and interpreting the resulting mass of data. Existing data analysis techniques have difficulty in handling multi-dimensional data. Multi-dimensional data has been a challenge for data analysis because of the inherent sparsity of the points.

A first step toward addressing this challenge is the use of clustering techniques, which is essential in the data mining process to reveal natural structures and identify interesting patterns in the underlying data. Cluster analysis is used to identify homogeneous and well-separated groups of objects in databases. The need to cluster large quantities of multi-dimensional data is widely recognized. It is a classical problem in the database, artificial intelligence, and theoretical literature, and plays an important role in many fields of business and science.
There are also a lot of approaches designed for outlier detection. In many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise. Thus, it is necessary to treat clusters and outliers as concepts of the same importance in data analysis.

It is well acknowledged that in the real world a large proportion of data has irrelevant features which may cause a reduction in the accuracy of some algorithms. High dimensional data sets continue to pose a challenge to clustering algorithms at a very fundamental level. One of the well known techniques for improving the data analysis performance is the method of dimension reduction which is often used in clustering, classification, and many other machine learning and data mining applications.

Many approaches have been proposed to index high-dimensional data sets for efficient querying. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data sets efficiently since it may greatly increase the amount of data accessed for a query.

In this dissertation, we study the problems mentioned above. We proposed a novel data preprocessing technique called shrinking which optimizes the inner structure of data inspired by Newton’s Universal Law of Gravitation in the real world. We then proposed a shrinking-based clustering algorithm for multi-dimensional data and extended the algorithm to the
dimension reduction field, resulting in a shrinking-based dimension reduction algorithm.

We also proposed a cluster-outlier iterative detection algorithm to detect the clusters and outliers in another perspective for noisy data sets. Apart from the shrinking-based data analysis and cluster-outlier interactive relationship exploration research, we designed a new indexing structure, ClusterTree+, for time-related high-dimensional data, which eliminates obsolete data dynamically and keeps the data in the most updated status so as to further promote the efficiency and effectiveness of data insertion, query and update.
# List of Figures

2.1.1 Examples for influence functions of Denclue. ........................................ 27

3.1.1 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. .... 45

3.2.1 An example of density span acquirement ............................................. 51

3.2.2 A data set with three clusters ......................................................... 54

3.3.1 Surrounding cells (in gray) of the cell $C$ (in black) .............................. 56

3.3.2 (a) Points $a, b, c,$ and $d$ are located near a vertex of the grid and are sepa-
rated 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. ........................................................................................................ 57
3.3.3 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. ........................................... 59

3.3.4 Repeated skeletonization. .................................................. 62

3.5.1 Two clusters with noisy points in between. .......................... 73

3.5.2 The cluster tree of a data set .............................. 75

3.6.1 The runtime and performance of Shrinking on wine data with different average movement threshold $T_{amv}$. ........................................... 78

3.6.2 Shrinking process on the data set $DS_1$ with cell size $0.1 \times 0.1$. (a) 2-dimensional data set $DS_1$, (b) the data set after the shrinking process .... 78

3.6.3 Running time with increasing dimensions .......................... 80

3.6.4 Running time with increasing data sizes ............................ 80

3.6.5 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. ........................................... 83
3.6.6 Testing result of OPTICS for Wine data (a) without shrinking preprocessing (b) after shrinking preprocessing ........................................ 88

3.6.7 Testing result of OPTICS for Pendigits data (a) without shrinking preprocessing (b) after shrinking preprocessing ........................................ 88

4.2.1 Histogram variance of Ecoli data on each dimension (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction ........................................ 99

4.3.1 The cut of the variance differences list on Wine data ........................................ 106

4.3.2 The cut of the variance differences list on Ecoli data ........................................ 107

4.3.3 The cut of the variance differences list on Pendigits data ........................................ 107

4.3.4 Testing result of OPTICS for Wine data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction ........................................ 108

4.3.5 Testing result of OPTICS for Ecoli data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction ........................................ 108

4.3.6 Testing result of OPTICS for Pendigits data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction ........................................ 109
4.3.7 Testing result of VizCluster for Wine data without shrinking-based dimension reduction ............................................. 109

4.3.8 Testing result of VizCluster for Wine data after shrinking-based dimension reduction ............................................. 110

4.3.9 Testing result of VizCluster for Ecoli data without shrinking-based dimension reduction ............................................. 110

4.3.10 Testing result of VizCluster for Ecoli data after shrinking-based dimension reduction ............................................. 111

5.2.1 Two clusters with same density but different compactness ................................................................. 123

5.2.2 Two clusters with same compactness but different orientations ............................................................ 123

5.2.3 Two cluster pairs with same average distance and minimum distance but different diversity ............................................. 126

5.2.4 Data groups associated with different medoids ................................................................. 131

5.3.1 Algorithm: COID ................................................................. 133

5.3.2 Proc: FindKMedoids ................................................................. 135
5.3.3 Ordered list of outlier subsets in the intermediate set based on $\frac{|x|}{d_{t(s)}}$ . . . . . . 136

5.3.4 Proc: DispatchDataPoints . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137

5.3.5 Proc: ClusterOrOutlier . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 141

5.3.6 Proc: MergeCluster . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144

5.3.7 Proc: ExchangeClusterAndOutlier . . . . . . . . . . . . . . . . . . . . . . . . . . . 144

5.4.1 The runtime and performance of COID on wine data with different values of Const1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146

5.4.2 Running time with increasing dimensions . . . . . . . . . . . . . . . . . . . . . . . 148

5.4.3 Running time with increasing data sizes . . . . . . . . . . . . . . . . . . . . . . . 149

6.2.1 An example of ClusterTree$^+$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161
List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6.1</td>
<td>Clustering results of our algorithm for Wine data</td>
<td>83</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Clustering result of CURE for Wine data as $\alpha=0.3$ and $r=30$</td>
<td>83</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Clustering result of our algorithm for Ecoli data</td>
<td>84</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Clustering result of CURE for Ecoli data as $\alpha=0.2$ and $r=30$</td>
<td>84</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Clustering result of our algorithm for Pendigits data. For the last four</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>columns, the corresponding relationship is two-to-one: two real clusters</td>
<td></td>
</tr>
<tr>
<td></td>
<td>correspond to one detected cluster.</td>
<td></td>
</tr>
<tr>
<td>3.6.6</td>
<td>Clustering result of CURE for Pendigits data as $\alpha=0.4$ and $r=50$</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>For the last four columns, the corresponding relationship is four-to-one:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>four real clusters correspond to one detected cluster.</td>
<td></td>
</tr>
</tbody>
</table>
3.6.7 Clustering result of CURE for Wine data after shrinking preprocessing as
\[ \alpha=0.3 \text{ and } r=30 \] 89

4.3.1 histogram variance differences of Wine data for a certain grid scale 105

4.3.2 histogram variance differences of Ecoli data for a certain grid scale 105

4.3.3 histogram variance differences of Pendigits data for a certain grid scale 106

5.4.1 Clustering result of CURE for Wine data as \( \alpha=0.3 \text{ and } r=10 \) 151

5.4.2 Clustering results of Shrinking algorithm for Wine data 151

5.4.3 Clustering results of COID algorithm for Wine data 152

5.4.4 Clustering result of CURE for Ecoli data as \( \alpha=0.8 \text{ and } r=20 \) 152

5.4.5 Clustering result of Shrinking algorithm for Ecoli data 153

5.4.6 Clustering result of COID algorithm for Ecoli data 153

5.4.7 Iteration of COID algorithm on clustering result of CURE algorithm on Wine data 153
5.4.8 Iteration of COID algorithm on clustering result of CURE algorithm on
Pendigits data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
## Contents

<table>
<thead>
<tr>
<th>Sections</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction to Data Mining</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Introduction to Cluster Analysis</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Clusters and clustering</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Proximity measurement</td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 Subspace Clustering</td>
<td>5</td>
</tr>
</tbody>
</table>
1.2.4 Cluster Evaluation .................................................. 6
1.3 Introduction to Outlier Detection ................................. 6
1.4 Introduction to Indexing .............................................. 7
1.5 Contributions of Current Research and Dissertation Layout .................................................. 7

2 Related Work ................................................................. 12

2.1 Clustering Multi-Dimensional Data .............................. 13

2.2 Cluster Validation ..................................................... 32

2.3 Outlier Detection ...................................................... 35

2.4 Indexing Multi-Dimensional Data ................................. 36

3 A Shrinking-Based Clustering Algorithm .......................... 41

3.1 Introduction ............................................................ 42

3.1.1 Proposed approach ................................................ 45

3.2 Choosing Grid Scales for the Shrinking-based Clustering Approach .................................................. 48
3.3 Data Shrinking ............................................. 54
  3.3.1 Space subdivision ..................................... 55
  3.3.2 Data movement in a single iteration ................. 58
  3.3.3 Termination of shrinking .............................. 61
  3.3.4 Time and space analysis .............................. 62

3.4 Cluster detection ........................................... 64

3.5 Cluster evaluation and selection ......................... 66
  3.5.1 Compactness graphing ................................. 67
  3.5.2 Grid-based compactness ............................... 73
  3.5.3 Evaluation and selection of multiscale clusters .... 75

3.6 Experiments ................................................. 76
  3.6.1 Experiments on 2D datasets ......................... 78
  3.6.2 Experiments on high-dimensional datasets .......... 79
  3.6.3 Experiments on real data ............................. 80
3.6.4 Experiments on how shrinking preprocessing improves clustering algorithms ... 88

3.7 Conclusion and discussion ... 90

4 A Shrinking-Based Dimension Reduction Approach ... 92

4.1 Introduction ... 93

4.1.1 Data shrinking ... 95

4.1.2 Proposed approach ... 97

4.2 Dimension Reduction ... 98

4.2.1 Core concept ... 98

4.2.2 Dimension reduction process ... 100

4.2.3 Time and space analysis ... 103

4.3 Experiments ... 103

4.3.1 Data sets ... 104

4.3.2 Shrinking-based dimension reduction approach on real data sets ... 105
4.3.3 Testing results of existing clustering algorithms . . . . . . . . . . . . 107

4.4 Conclusion and discussion . . . . . . . . . . . . . . . . . . . . . . . . . . 112

5 A Cluster-Outlier Iterative Detection Approach 113

5.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114

5.2 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

5.2.1 Distance metric selection . . . . . . . . . . . . . . . . . . . . . . . . 119

5.2.2 The compactness of a cluster . . . . . . . . . . . . . . . . . . . . . . 120

5.2.3 The diversities of data groups . . . . . . . . . . . . . . . . . . . . . . 122

5.2.4 The qualities of data groups . . . . . . . . . . . . . . . . . . . . . . . 129

5.3 Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 131

5.3.1 Initialization phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

5.3.2 Iterative phase . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138

5.3.3 Time and space analysis . . . . . . . . . . . . . . . . . . . . . . . . . 143
5.4 Experiments ................................................. 145

5.4.1 Experiments on high-dimensional data sets ............... 147

5.4.2 Data sets .................................................. 148

5.4.3 Algorithms .............................................. 149

5.4.4 Using COID to improve other algorithms ................. 154

5.5 Conclusion and discussion .................................. 156

6 Dynamic Clustering and Indexing of Multi-Dimensional Data 157

6.1 Introduction ............................................... 158

6.2 ClusterTree$^+$ and its Construction ......................... 161

6.2.1 Construction .......................................... 165

6.3 Processing of the ClusterTree$^+$ .............................. 165

6.3.1 Insertion .............................................. 166

6.3.2 Query ................................................. 167
6.3.3 Deletion ................................................................. 168

6.4 Conclusion .............................................................. 171

7 Conclusion and Future Work .............................................. 172

7.1 Conclusion .............................................................. 173

7.1.1 Shrinking-based data preprocessing and clustering algorithms . . . 174

7.1.2 Shrinking-based dimension reduction algorithms ....................... 175

7.1.3 Cluster and Outlier Iterative Detection .................................... 177

7.1.4 indexing time-related multi-dimensional data sets ....................... 178

7.2 Future Work ............................................................ 179

7.2.1 Cluster and outlier iterative detection in subspace ....................... 180

7.2.2 dynamically insertion for indexing structure of time-related multi-
dimensional data ............................................................. 184

7.2.3 combination of fuzzy clustering and shrinking-based data analysis
approaches. ................................................................. 190
Chapter 1

Introduction

1.1 Introduction to Data Mining

Data mining is the task of discovering interesting patterns from large amounts of data where the data can be stored in databases, data warehouses, or other information repositories. It is also popularly referred to as knowledge discovery in databases (KDD). Data mining involves an integration of techniques from multiple disciplines such as database technology, statistics, machine learning, high-performance computing, pattern recognition, neural networks, data visualization, information retrieval, etc.

The architecture of a typical data mining system may have the following major components[47]:
database, data warehouse, or other information repository; their server which is responsible for fetching the relevant data based on the user’s data mining request; knowledge base which is used to guide the search, or evaluate the interestingness of resulting patterns; data mining engine which consists of a set of functional modules for tasks; pattern evaluation module which interacts with the data mining modules so as to focus the search towards interesting patterns; and graphical user interface which communicates between users and the data mining system, allowing the user interaction with system.

Data mining tasks have the following categories:

- Class description. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms.

- Association analysis. It is the discovery of association rules showing attribute-value conditions that occur frequently together in a given set of data.

- Classification. It is the process of finding a set of models that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data, and can be represented in forms like classification rules, decision trees, etc.

- Cluster analysis. Clustering analyzes data objects without consulting a known class
label. In general, the class labels are not present in the training data simply because they are not known to begin with. The objects are clustered or grouped based on the principle of maximizing the intra-class similarity and minimizing the inter-class similarity.

- Outlier analysis. Outliers are data objects that do not comply with the general behavior of model of the data. Outliers may be detected using statistical tests or using distance measures.

- Evolution analysis. It describes and models trends for objects whose behaviors changes over time. It normally includes time-series data analysis, sequence or periodicity pattern matching, and similarity-based data analysis.

1.2 Introduction to Cluster Analysis

1.2.1 Clusters and clustering

Cluster analysis is one of the classical topics in the data mining field and is often regarded as the first step toward exciting knowledge discovery. Clustering is the process of grouping data objects into a set of disjoint classes, called clusters, so that objects within a class have high similarity to each other, while objects in separate classes are more dissimilar.
Clustering is an example of unsupervised classification. “Classification” refers to a procedure that assigns data objects to a set of classes. “Unsupervised” means that clustering does not rely on predefined classes and training examples while classifying the data objects. Thus, clustering is distinguished from pattern recognition or the areas of statistics known as discriminant analysis and decision analysis, which seek to find rules for classifying objects from a given set of pre-classified objects. Existing clustering algorithms can be broadly classified into four types [46]: partitioning [43, 52, 64], hierarchical [93, 35, 34], grid-based [83, 76, 6], and density-based [27, 42, 8] algorithms. Partitioning algorithms construct a partition of a database of \( n \) objects into a set of \( K \) clusters, where \( K \) is an input parameter. Hierarchical algorithms create a hierarchical decomposition of the given data set of data objects. The hierarchical decomposition is represented by a tree structure, called dendrogram. Grid-based algorithms quantize the space into a finite number of grids and perform all operations on this quantized space. These approaches have the advantage of fast processing time independent of the data set size and are dependent only on the number of segments in each dimension in the 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. Density-based approaches can also filter out outliers.
1.2.2 Proximity measurement

Many clustering algorithms rely on some proximity measure to evaluate the distance or similarity between two data objects. A data object $O$ can be formalized as numerical vectors $\vec{O} = \{o_d|1 \leq d \leq p\}$, where $o_d$ is the value of the $d$th attribute for $O$ and $p$ is the number of attributes. The proximity between two objects $O_i$ and $O_j$ is measured by a proximity function of corresponding vectors $\vec{O}_i$ and $\vec{O}_j$.

*Euclidean distance* is one of the most commonly-used methods to measure the distance between two data objects. The distance between objects $O_i$ and $O_j$ in $p$-dimensional space is defined as:

\[
(1.2.1) \quad \text{Euclidean}(O_i, O_j) = \sqrt{\sum_{d=1}^{p} (o_{id} - o_{jd})^2}.
\]

However, in some cases such as analyzing gene expression data, the overall shapes of object patterns are of greater interest than the individual magnitudes of each feature. Euclidean distance does not score well for shifting or scaled patterns [84].

1.2.3 Subspace Clustering

Clustering in high dimensional space is often problematic as theoretical results [16] questioned the meaning of closest matching in high dimensional spaces. Recent research work [2,
4, 6, 22, 44] has focused on discovering clusters embedded in the subspaces of a high dimensional data set. This problem is known as *subspace clustering*. Based on the measure of similarity, there are two categories of clustering model. The first category is *pattern-based clustering*. The second category is *distance-based clustering*.

### 1.2.4 Cluster Evaluation

A lot of approaches [21, 59, 60, 61] have been proposed for evaluating the results of a clustering algorithm. 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.

### 1.3 Introduction to Outlier Detection

Outlier detection is concerned with discovering the exceptional behaviors of certain objects. It is an important branch in the field of data mining with numerous applications, including credit card fraud detection, discovery of criminal activities, discovery of computer intrusion, and etc. In some sense it is at least as significant as cluster detection. There
CHAPTER 1. INTRODUCTION

are numerous studies on outlier detection.

1.4 Introduction to Indexing

Many approaches have been proposed to index high-dimensional data sets for efficient querying. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data sets efficiently since it may greatly increase the amount of data accessed for a query.

1.5 Contributions of Current Research and Dissertation

Layout

In this dissertation, we focus on dynamic data mining for multi-dimensional data. We start with a data preprocessing approach called shrinking, then we propose a shrinking-based clustering algorithm. We then discuss a cluster-outlier iterative detection approach, as well as an indexing structure for time-related data sets.
Existing data analysis techniques have difficulty in handling multi-dimensional data. Multi-dimensional data has been a challenge for data analysis because of the inherent sparsity of the points. Various clustering methods have been applied to identify clusters for multi-dimensional data. However, there are difficulties for high-dimensional clustering due to the characteristics of the data such as the various densities of the clusters, irregular shapes of the clusters, etc. In Chapter 3, we study the problem of clustering multi-dimensional data, and make the following contributions:

- We present a novel data preprocessing technique which optimizes the inherent characteristic of distribution 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. The data-shrinking steps can be used as a preprocessing procedure for any cluster-detection algorithm. We will demonstrate how the data shrinking 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.

It is well acknowledged that in the real world a large proportion of data has irrelevant features which may cause a reduction in the accuracy of some algorithms. High dimensional data sets continue to pose a challenge to clustering algorithms at a very fundamental level. One of the well known techniques for improving the data analysis performance is the method of dimension reduction[6, 4, 73] in which data is transformed to a lower dimensional space while preserving the major information it carries, so that further processing can be simplified without compromising the quality of the final results. Dimension reduction is often used in clustering, classification, and many other machine learning and data mining applications. In Chapter 4, we study the problem of dimension reduction, and make the following contributions:

- We extend the shrinking data preprocessing approach and the clustering algorithm.
- We propose a dimension reduction approach which tends to solve the dimension reduction problem from a new perspective.

Nowadays many data mining algorithms focus on clustering methods. There are also a lot
of approaches designed for outlier detection. We observe that, in many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise. Thus, it is necessary to treat clusters and outliers as concepts of the same importance in data analysis. In Chapter 5, we study the problem of clustering and outlier detection simultaneously, and make the following contribution:

- We explore a new way to define the distance between two clusters, the distance between two outliers, and the one between a cluster and an outlier.

- We apply some novel formula to define the qualities of the clusters, the outliers, and the division of the whole data set.

- We propose an algorithm to detect clusters and outliers simultaneously. Clusters are detected and adjusted according to the intra-relationship within clusters and the inter-relationship between clusters and outliers, and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached.

Large volumes of data with high dimensionality are being generated in many fields. Most existing indexing techniques degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data set efficiently since it may greatly increase the amount of data accessed for a query. In Chapter
6, we study the indexing structures for multi-dimensional data from the conceptual and theoretical point of view, and make the following contributions:

- We extend the ClusterTree, enhancing the indexing structure with time features.

- We improve the performance of data insertion, query and update on time-related multi-dimensional data based on the time-related indexing structure.
Chapter 2

Related Work

This dissertation is closely related to dynamic data mining, more specifically, to clustering, outlier detection and indexing for multi-dimensional data. As we introduced in Chapter 1, there are a lot of algorithms for clustering such as partitioning, hierarchical, grid-based, and density-based clustering algorithms, as well as subspace clustering algorithms. There are also many outlier detection algorithms and indexing algorithms such as LOF, R-Tree, R’-tree, SS-Tree, SS+ -Tree, SR-Tree, etc. In section 2.1, we will give a brief survey for clustering algorithms, and introduce the basic concepts of different types of clustering algorithms. In section 2.2, we will introduce cluster validation approaches. In section 2.3 we will discuss various outlier detection algorithms. In section 2.4, some indexing approaches will be introduced.
2.1 Clustering Multi-Dimensional Data

Cluster analysis seeks to partition a given data set into groups based on specified features so that the data points within a group are more similar to each other than the points in different groups. A very rich literature on cluster analysis has developed over the past three decades. It is a classical problem in the database, artificial intelligence, and theoretical literature, and plays an important role in many fields of business and science. The basic steps in the development of a clustering process can be summarized as feature selection, application of a clustering algorithm, validation of results, and interpretation of the results. Among these steps, the clustering algorithm and validation of the results are especially critical, and many methods have been proposed in the literature for these two steps.

Existing clustering algorithms can be broadly classified into four types [46]: partitioning [43, 52, 64], hierarchical [93, 35, 34], grid-based [83, 76, 6], and density-based [27, 42, 8] algorithms.

Partitioning Algorithms

Partitioning algorithms divide a data set into several mutually-exclusive subsets based on certain clustering assumptions (e.g., there are \( k \) clusters in the data set) and optimization criteria (e.g., minimize the sum of distances between objects and their cluster centroids).
In general, 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. Some popular partitioning algorithms include: the K-means algorithm and its derivatives [40, 43, 67, 78, 80], the Self-Organizing Map (SOM) and its extensions [39, 55, 79, 81], graph-based algorithms [12, 38, 75, 87] and model-based algorithms [31, 32, 63, 89].

**K-means and its Derivatives.** The K-means algorithm [43] is a typical partitioning clustering method. Given a pre-specified number \( K \), the algorithm partitions the data set into \( K \) disjoint subsets which optimize the following objective function:

\[
E = \sum_{i=1}^{K} \sum_{O \in C_i} |O - \mu_i|^2.
\]

Here, \( O \) is a data object in cluster \( C_i \) and \( \mu_i \) is the centroid (mean of objects) of \( C_i \). Thus, the objective function \( E \) tries to minimize the sum of the squared distances of objects from their cluster centers.

The K-means algorithm is simple and fast. The time complexity of K-means is \( O(l \times k \times n) \), where \( l \) is the number of iterations and \( k \) is the number of clusters. However, the K-means algorithm also has several drawbacks. First, the number of the clusters in a data set is usually unknown in advance. To detect the optimal number of clusters, users usually run the algorithms repeatedly with different values of \( k \) and compare the clustering results. For a large data set this extensive parameter fine-tuning process may not be practical. Second,
real data sets typically contain a huge amount of noise; however, the K-means algorithm forces each data point into a cluster, which may cause the algorithm to be sensitive to noise [77, 78].

Recently, several new algorithms [40, 67, 78] have been proposed to overcome the drawbacks of the K-means algorithm. We call them derivatives of the K-means algorithm, since in essence, they also pursue minimizing the overall divergence of objects from their cluster centers. One common feature of those algorithms is that, they use some thresholds to control the coherence of clusters. For example, Ralf-Herwig et al. [67] introduce two parameters $\gamma$ and $\rho$, where $\gamma$ is the maximal similarity between two separate cluster centroids, and $\rho$ corresponds to the minimal similarity between a data point and its cluster centroid.

In [40], the clusters are constrained to have a diameter no larger than $d$. Motivated by [40], Smet et al. [78] propose a more efficient algorithm Adapt Cluster. Data object $x$ will be assigned to cluster $c$ if the assignment has a higher probability than threshold $S$. Compared with [67] and [40], the cluster model in [78] is more adaptive to various data structures, since the clusters are not rigidly bounded by the radius or diameter threshold.

The clustering process of the above algorithms turns out to be extracting all the clusters with qualified coherence from the data set. Therefore, users do not need to input the number of clusters. In addition, with the coherence control, outliers may only end up with trivial clusters, i.e., clusters with very few members.
**SOM and its Extensions.** The Self-Organizing Map (SOM) was developed by Kohonen [55], on the basis of a single layered neural network. The data objects are presented at the input, and the output neurons are organized with a simple neighborhood structure such as a two-dimensional \( p \times q \) grid. Each neuron of the neural network is associated with a reference vector, and each data point is “mapped” to the neuron with the “closest” reference vector. In the process of running the algorithm, each data object acts as a training sample which directs the movement of the reference vectors towards the denser areas of the input vector space, so that those reference vectors are trained to fit the distributions of the input data set. When the training is complete, clusters are identified by mapping all data points to the output neurons.

One of the remarkable features of SOM is that it allows users to impose partial structure on the clusters, and arranges similar patterns as neighbors in the output neuron map. This feature facilitates easy visualization and interpretation of the clusters.

However, similar to the K-means algorithm, SOM also requires a user to specify the number of clusters, which is typically unknown in advance in the case of read data sets. Moreover, as pointed out in [39], if the data set is abundant with irrelevant data points, such as genes with invariant patterns, SOM will produce an output where irrelevant data points will populate the vast majority of clusters, while most interesting patterns will be missed since they are collapsed into only a few clusters.
Recently, several new algorithms [39, 62, 81] have been proposed based on the SOM algorithm. Those algorithms can automatically determine the number of clusters and dynamically adapt the map structure to the data distribution. For example, Herrero et al. [39] extend the SOM by a binary tree structure. At first, the tree only contains a root node connecting two neurons. After a training process similar to that of the SOM algorithm, the data set is segregated into two subsets. Then the neuron with less coherence is split in two new neurons. This process is repeated level by level, until all the neurons in the tree satisfy some coherence threshold. Other examples of SOM extensions are Fuzzy Adaptive Resonance Theory (Fuzzy ART) [81] and supervised Network Self-Organized Map (sNet SOM) [62]. In general, they provide some approaches to measure the coherence of a neuron (e.g. vigilance criterion in [81] and grow parameter in [62]). The output map is adjusted by splitting existing neurons or adding new neurons into the map, until the coherence of each neuron in the map satisfies a user-specified threshold.

SOM is an efficient and robust clustering technique. Hierarchical structure can also be built based on SOM (e.g. SOTA). Moreover, by systematically controlling the split of neurons, SOM can easily adapt to the local structures of the data set. However, the current approaches control the splitting process by some coherence threshold, which is hard for users to specify.

**Graph-theoretical approaches.** Given a data set $X$, we can construct a *proximity matrix $P$, \[ P_{ij} = \begin{cases} 1 & \text{if } X_i \text{ is close to } X_j \text{ for some chosen proximity measure} \\ 0 & \text{otherwise} \end{cases} \]
where $P[i, j] = \text{proximity}(O_i, O_j)$, and a weighted graph $G(V, E)$, called a proximity graph, where each data point corresponds to a vertex. For some clustering methods, each pair of objects is connected by an edge with weight assigned according to the proximity value between the objects [75, 86]. For other methods, proximity is mapped only to either 0 or 1 on the basis of some threshold, and edges only exist between objects $i$ and $j$, where $P[i, j]$ equals 1 [12, 38]. Graph-theoretical clustering techniques are explicitly presented in terms of a graph, thus converting the problem of clustering a dataset into such graph theoretical problems as finding minimum cut or maximal cliques in the proximity graph $G$.

HCS and CLICK. Hartuv et al. [38] propose an algorithm HCS (for Highly Connected Subgraph), which recursively splits the weighted graph $G$ into a set of highly connected components along the minimum cut. Each highly connected component is considered as a cluster. Motivated by HCS, Shamir et al. present the algorithm CLICK (for CLuster Identification via Connectivity Kernels) in [75]. CLICK makes the probabilistic assumption that after standardization, pair-wise similarity values between elements (no matter they are in the same cluster or not) are normally distributed. Under this assumption, the weight $\omega_{ij}$ of an edge $(i, j)$ is defined as the probability that vertices $i$ and $j$ are in the same cluster. The clustering process of CLICK iteratively finds the minimum cut in the proximity graph and recursively splits the data set into a set of connected components from the minimum cut. CLICK also takes two post-pruning steps to refine the cluster results. The adoption step handles the remaining singletons and updates the current clusters, while the merging
step iteratively merges two clusters with similarity exceeding a predefined threshold.

CAST. Ben-Dor et al. [12] introduced the idea of a corrupted clique graph data model. The input data set is assumed to come from the underlying cluster structure by “contamination” with random errors caused by the complex process of gene expression measurement. Specifically, it is assumed that the true clusters of the data points can be represented by a clique graph $H$, which is a disjoint union of complete sub-graphs with each clique corresponding to a cluster. The similarity graph $G$ is derived from $H$ by flipping each edge/non-edge with probability $\alpha$. Therefore, clustering a dataset is equivalent to identifying the original clique graph $H$ from the corrupted version $G$ with as few flips (errors) as possible.

In [12], Ben-Dor et al. presented both a theoretical algorithm and a practical heuristic called CAST (Cluster Affinity Search Technique). CAST takes as input a real, symmetric, n-by-n similarity matrix $S$ ($S(i, j) \in [0, 1]$ and an affinity threshold $t$. The algorithm searches the clusters one at a time. The currently searched cluster is denoted by $C_{open}$. Each element $x$ has an affinity value $a(x)$ with respect to $C_{open}$ as $a(x) = \sum_{y \in C_{open}} S(x, y)$. An element $x$ has a high affinity value if it satisfies $a(x) \geq t|C_{open}|$; otherwise, $x$ has a low affinity value. CAST alternates between adding high-affinity elements to the current cluster, and removing low-affinity elements from it. When the process stabilizes, $C_{open}$ is considered a complete cluster, and this process continues with each new cluster until all elements have been assigned to a cluster.
CHAPTER 2. RELATED WORK

The affinity threshold $t$ of the CAST algorithm is actually the average of pairwise similarities within a cluster. CAST specifies the desired cluster quality through $t$ and applies a heuristic searching process to identify qualified clusters one at a time. Therefore, CAST does not depend on a user-defined number of clusters and deals with outliers effectively. Nevertheless, CAST has the usual difficulty of determining a “good” value for the global parameter $t$.

MST. In [87], Xu et al. first generate a Minimum Spanning Tree (MST) from the weighted graph $G$ of data set $X$. By removing $(K - 1)$ edges from the generated MST, the data set is partitioned into $K$ clusters. Three alternate algorithms, i.e., clustering through removing long MST-edges, an iterative clustering, and a globally optimal clustering algorithm are presented to determine the edges to be removed.

**Hierarchical Algorithms**

In contrast to partitioning clustering, which attempts to directly decompose the data set into a set of disjoint clusters, hierarchical clustering generates a hierarchical series of nested clusters which can be graphically represented by a tree, called *dendrogram*. The branches of a dendrogram not only record the formation of the clusters but also indicate the similarity between the clusters. By cutting the dendrogram at some level, we can obtain a specified number of clusters. By reordering the objects such that the branches of the corresponding
dendrogram do not cross, the data set can be arranged with similar objects placed together. Hierarchical clustering algorithms can be further divided into agglomerative approaches and divisive approaches based on how the hierarchical dendrogram is formed.

**Agglomerative algorithms.** Agglomerative algorithms (bottom-up approach) initially regard each data object as an individual cluster, and at each step, merge the closest pair of clusters until all the groups are merged into one cluster. Different measures of cluster proximity, such as single link, complete link and minimum-variance [45, 52], derive various merge strategies.

However, as pointed out in previous studies [9, 79], traditional agglomerative clustering algorithms may not be robust to noise. They often make the decisions of merging based on local information and never trace back, i.e., any “bad” decisions made in the initial steps may never get corrected later. In addition, hierarchical clustering only provides a tree structure (called *dendrogram*). There is no standard to decide where to cut the dendrogram to derive clusters. It is hard for users to manually inspect the whole tree.

To make the traditional agglomerative method more robust to the noise, Šášík et al. [72] propose a novel approach called *percolation clustering*. In essence, percolation clustering adopts a statistical bootstrap method to merge two data objects (or two subsets of data objects) when they are significantly coherent with each other. In [9], Bar-Joseph et al. replace the traditional binary hierarchical tree with a $k$-ary tree, where each non-leaf node
is allowed to have at most $k$ children. A heuristic algorithm is also presented to construct the $k$-ary tree, reduce susceptibility to noise and generate an optimal order for the leaf nodes. The two approaches above make the derived hierarchical tree more robust, however, neither of them indicate how to cut the dendrogram to obtain meaningful clusters, either.

Seo et al. [74] develop an interactive tool, *Hierarchical Clustering Explorer (HCE)*, to help users derive clusters from the dendrogram. To be specific, HCE visualizes the dendrogram by setting the distance from the root to an internal node $N$ according to the coherence between the two children $N_1$ and $N_2$ of $N$. That is, the more coherent are $N_1$ and $N_2$, the more distant is $N$ from the root. A user can select how to cut the dendrogram in horizontal by dragging the “minimum similarity bar”. However, the system of HCE is only a visualization tool that facilitates the inspection of the dendrogram. In other words, it does not provide any hint to where the dendrogram should be cut.

**Divisive Hierarchical Algorithms.** Divisive algorithms (i.e., top-down approaches) start with one cluster containing all the data objects. They iteratively split clusters until each cluster contains only one data object or certain stop criterion is met. For divisive approaches, the essential problem is to decide how to split clusters at each step.

For example, *Super-Paramagnetic Clustering (SPC)* was proposed by Blatt et al. [17] based on the physical properties of an inhomogeneous ferromagnetic model. *SPC* first transform the data set into a distance matrix between the $N$ data objects that are to be clustered. A
distance graph is then constructed whose vertices are the data objects and edges identifying neighboring objects. Two objects $O_i$ and $O_j$ are called neighbors if they satisfy the $K$-mutual-neighbor criterion, i.e., iff $O_j$ is one of the $K$ nearest objects to $O_i$ and vice versa. Each edge in the constructed graph is associated with a weight $J_{ij} > 0$, which decreases as the distance between objects $O_i$ and $O_j$ increases.

The clustering process of $SPC$ is equivalent to partitioning the weighted graph. Cluster indices play the role of the states of Potts spins assigned to each vertex (i.e. to each original data object). Two neighboring spins are interacting ferromagnetically with strength $J_{ij}$. This Potts ferromagnet is simulated at a sequence of temperatures $T$. At very low temperatures all objects belong to a single cluster and as $T$ is increased, clusters break into smaller ones until at high enough temperatures each object forms its own cluster. The clusters found at all temperatures form a dendrogram.

In summary, hierarchical clustering provides a natural way to graphically represent the data set. The graphic representation allows users a thorough inspection of the whole data set so that the users can obtain an initial impression of the distribution of data. However, the conventional agglomerative approach suffers from a lack of robustness [79], i.e., a small perturbation of the data set may greatly change the structure of the hierarchical dendrogram. Another drawback of the hierarchical approach is its high computational complexity. To construct a “complete” dendrogram (where each leaf node corresponds to one data
object, and the root node corresponds to the whole data set), the clustering process should take \( \frac{n^2 - n}{2} \) merging (or splitting) steps. The time complexity for a typical agglomerative hierarchical algorithm is \( O(n^2 \log n) \) [46]. Furthermore, for both agglomerative and divisive approaches, the “greedy” nature of hierarchical clustering prevents the refinement of the previous clustering. If a “bad” decision is made in the initial steps, it can never be corrected in the following steps.

**Grid-based Algorithms**

Grid-based algorithms quantize the space into a finite number of grids and perform all operations on this quantized space. These approaches have the advantage of fast processing time independent of the data set size and are dependent only on the number of segments in each dimension in the quantized space. Traditional grid-based algorithms include Wavecluster[76], STING[6], etc.

WaveCluster is a novel clustering approach based on wavelet transforms, which satisfies all the above requirements. Using the multiresolution property of wavelet transforms, it can effectively identify arbitrarily shaped clusters at different degrees of detail. WaveCluster is highly efficient in terms of time complexity. Experimental results on very large data sets show the efficiency and effectiveness of the approach compared to the other recent clustering methods.
CHAPTER 2. RELATED WORK

STING is a hierarchical statistical information grid based approach for spatial data mining to reduce the cost. The idea is to capture statistical information associated with spatial grids in such a manner that whole classes of queries and clustering problems can be answered without recourse to the individual objects. In theory, and confirmed by empirical studies, this approach performs well especially when the data set is very large.

Density-based Approaches

Density-based approaches describe the distribution of a given data set by the “density” of data objects. The clustering process involves a search of the “dense areas” in the object space [28].

DBSCAN. In [28], Ester et al. introduced the DBSCAN algorithm which relies on a density-based notion of clusters. To measure the “density” of data objects, DBSCAN first defines the Eps-neighborhood of an object $p$ as a set of objects $N_{\text{eps}}(p)$ such that the distance between $p$ and each object $q$ in $N_{\text{eps}}(p)$ is smaller than a user-specified threshold $Eps$. Intuitively, an object $p$ has a “high” density (and thus called a core object) if $N_{\text{eps}}(p) \geq \text{MinPts}$, where $\text{MinPts}$ is a user-specified threshold. An object $p$ is called density-connected to an object $q$ w.r.t. $Eps$ and $\text{MinPts}$ if there is an object $o$ such that both $p$ and $q$ are density-reachable from $o$. The density-reachability from $q$ to $p$ w.r.t. $Eps$ and $\text{MinPts}$ is described by two situations. First, $p$ is called directly density-reachable from $q$ if $p \in N_{\text{eps}}(q)$ and
$q$ is a core object. A general definition is given as follows: $p$ is density-reachable from $q$ if there is a chain of objects $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that $p_{i+1}$ is directly density-reachable from $p_i$ ($1 \leq i \leq n - 1$). Given the two thresholds $Eps$ and $MinPts$, DBSCAN defines a cluster $C$ as a non-empty set of data objects satisfying the following two conditions: (1) $\forall p, q$: if $p \in C$ and $q$ is density-reachable from $p$, then $q \in C$; and (2) $\forall p, q \in C$, $p$ is density-connected to $q$. Naturally, the noise is the set of objects which do not belong to any clusters. The clustering process of DBSCAN scans the data set only once and reports all the clusters and noise.

OPTICS. While DBSCAN is able to discover clusters with arbitrary shape and has good efficiency on large data sets, the algorithm is very sensitive to the input parameters. It may generate very different clustering results for slightly different parameter settings [8]. On the basis of DBSCAN, Ankerst et al. [8] introduced the algorithm OPTICS, which does not generate clusters explicitly, but instead creates an ordering of the data objects and illustrates the cluster structure of the data set. However, when applied to a highly-connected data set, OPTICS may enter another dense area through “intermediate” data objects before the visiting of the current dense area is completed. Consequently, the data objects in the same cluster may not be accommodated consecutively in the order.

Denclue. Different from the local density measures by DBSCAN and OPTICS, Denclue [42] measures object density from a global perspective: data objects are assumed to “influence”
each other, and the density of a data object is the sum of influence functions from all data objects in the data set. By assigning different influence functions, Denclue is a generalization of many partition-based, hierarchical, and density-based clustering methods. Figure 2.1.1 illustrates the effect of different influence functions. Denclue is robust to large amount of noise and works well for high-dimensional data sets. However, it outputs all clusters at the same level. Therefore, it cannot support an exploration of hierarchical cluster structures which exploits users’ domain knowledge.

**Subspace Clustering**

Clustering in high dimensional space is often problematic as theoretical results [16] questioned the meaning of closest matching in high dimensional spaces. Recent research work [2, 4, 6, 22, 44] has focused on discovering clusters embedded in the subspaces of a high dimensional data set. This problem is known as **subspace clustering**. Based on the measure
of similarity, there are two categories of clustering model.

The first category is pattern-based clustering. A pattern-based cluster\[20, 58, 66, 84, 88\] consists of a subset of objects $O$ and a subset of attributes $A$ such that $O$ exhibit coherent patterns on $A$. In pattern-based clustering, the subsets of attributes for various pattern-based clusters can be different. Two pattern-based clusters can share some common objects and attributes, and some objects may not belong to any pattern-based cluster.

Cheng et al. \[20\] introduced the bicluster concept to model a block, along with a score called the mean-squared residue to measure the coherence of genes and conditions in the block. Given a subset of objects $I$ and a subset of attributes $J$, the coherence of the submatrix $(I, J)$ is measured by the mean squared residue score.

\[
(2.1.2) \quad r_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} (a_{ij} - a_{ij} - a_{lj} + a_{IJ})^2,
\]

where

\[
(2.1.3) \quad a_{ij} = \frac{1}{|J|} \sum_{j \in J} a_{ij}, \quad a_{lj} = \frac{1}{|I|} \sum_{i \in I} a_{ij}, \quad \text{and} \quad a_{IJ} = \frac{1}{|I||J|} \sum_{i \in I, j \in J} a_{ij}
\]

are the average value of row $i$, the average value of column $j$, and the average value of the submatrix $(I, J)$, respectively. A submatrix is called a $\delta$-bicluster if $H(G', S') \leq \delta$ for some $\delta > 0$. A low mean-squared residue score together with a large variation from the constant suggest a good criterion for identifying a block.

Since the problem of finding a minimum set of biclusters to cover all the elements in a data
matrix has been shown to be NP-hard, a greedy method which provides an approximation of the optimal solution and reduces the complexity to polynomial-time has been introduced in [20]. To find a bicluster, the score $H$ is computed for each possible row/column addition/deletion, and the action that decreases $H$ the most is applied. If no action will decrease $H$ or if $H \leq \delta$, a bicluster is returned. However, this algorithm (the brute-force deletion and addition of rows/columns) requires computational time $O((n+m) \cdot mn)$, where $n$ and $m$ are the number of genes and samples, respectively, and it is time-consuming when dealing with a large gene expression data sets. A more efficient algorithm based on multiple row/column addition/deletion (the biclustering algorithm) with time-complexity $O(mn)$ was also proposed in [20]. After one bi-cluster is identified, the elements in the corresponding sub-matrix are replaced (masked) by random numbers. The bi-clusters are successively extracted from the raw data matrix until a pre-specified number of clusters have been identified. However, the biclustering algorithm also has several drawbacks. First, the algorithm stops when a pre-specified number of clusters have been identified. To cover the majority of elements in the data matrix, the specified number is usually large. However, the biclustering algorithm does not guarantee that the biclusters identified earlier will be of superior quality to those identified later, which adds to the difficulty of the interpretation of the resulting clusters. Second, biclustering “mask” the identified biclusters with random numbers, preventing the identification of overlapping biclusters.

$\delta$-Cluster. Yang at al. [88] present a pattern-based clustering method named “$\delta$-clusters”
to capture $K$ embedded subspace clusters simultaneously. They use average $residue$ across every entry in the sub-matrix to measure the coherence within a submatrix. A heuristic move-based method called FLOC (FLexible Overlapped Clustering) is applied to search $K$ embedded subspace clusters. FLOC starts with $K$ randomly selected sub-matrices as the subspace clusters, then iteratively tries to add/remove each row/column into/out of the subspace clusters to lower the $residue$ value, until a local minimum $residue$ value is reached.

The time complexity of the $\delta$-clusters algorithm is $O((n + m) \times n \times m \times k \times l)$, where $k$ is the number of clusters and $l$ is the number of iterations. $\delta$-clusters algorithm also requires the number of the clusters be pre-specified. The advantage of the “$\delta$-clusters” approach is that it is robust to missing values since the $residue$ of a submatrix only computed by existing values. “$\delta$-clusters” can also detect overlapping embedded subspace clusters.

In [84], Wang et al. propose the model of $\delta$-pCluster. To constrain the coherence of patterns in a cluster $C$, $\delta – pCluster$ requires the change of differences of any two data objects $x$, $y$ in $C$ on any two attributes $a$, $b$ in $C$ should be smaller than a threshold $\delta$. To be formal, let $\emptyset$ be a subset of objects in the database ($\emptyset \subseteq D$), and let $\mathcal{T}$ be a subset of a subset of attributes ($\mathcal{T} \subseteq A$). Pair $(\emptyset, \mathcal{T})$ specifies a submatrix. Given $x, y \in \emptyset$, and $a, b \in \mathcal{T}$, the $pScore$ of the $2 \times 2$ matrix is defined as

$$
(2.1.4) \quad pScore \left( \begin{bmatrix} d_{xa} & d_{xb} \\ d_{ya} & d_{yb} \end{bmatrix} \right) = |(d_{xa} - d_{xb}) - (d_{ya} - d_{yb})|
$$

Pair $(\emptyset, \mathcal{T})$ forms a $\delta – pCluster$ if for any $2 \times 2$ submatrix $X$ in $(\emptyset, \mathcal{T})$, we have $pScore(X) \leq$
δ for some δ > 0. In a recent study [66], Pei et al. develop MaPle, an efficient algorithm to mine the complete set of non-redundant pattern-based clusters.

**OP-Cluster.** In [58], Liu et al. present the model of OP-Cluster. Under this model, the coherence within a cluster is constrained by the relative order of the attribute values of the objects within the cluster. To be specific, an object \( O \) shows the “UP” pattern on a permutation \( A' = \{a_{i1}, \ldots, a_{in}\} \) of attributes \( A = \{a_1, \ldots, a_n\} \) if the attribute values \( \langle d_{i1}, \ldots, d_{in}\rangle \) of \( O \) on \( A' \) is in a “non-decreasing” order. Let \( \emptyset \) be a subset of objects in the database \( (\emptyset \subseteq D) \), and let \( T \) be a subset of a subset of attributes \( (T \subseteq A) \). Pair \( (\emptyset, T) \) forms an Order Preserving Cluster (OP-Cluster) if there exists a permutation of attributes in \( T \), on which every object in \( \emptyset \) shows the “UP” pattern. In practice, the requirement of “non-decreasing order” is often relaxed by a group function. In [58], an efficient algorithm, OPC-Tree, is proposed to find the OP-Clusters. Both MaPle and OPC-Tree guarantee to find the complete set of the pattern-based clusters in the data set.

The second category is distance-based clustering. In this category, one of the well known subspace clustering algorithms is CLIQUE [6]. CLIQUE is a density and grid based clustering method. The PROCLUS [4] and the ORCLUS [2] algorithms find projected clusters based on representative cluster centers in a set of cluster dimensions. Another interesting approach, Fascicles [44], finds subsets of data that share similar values in a subset of dimensions.
2.2 Cluster Validation

Different clustering algorithms, or even a single clustering algorithm using different parameters, generally result in different sets of clusters. Therefore, it is important to compare various clustering results and select the one that best fits the “true” data distribution. Cluster validation is the process of assessing the quality and reliability of the cluster sets derived from various clustering processes.

A lot of approaches [21, 59, 60, 61] have been proposed for evaluating the results of a clustering algorithm. Various methods have been designed to assess the quality and reliability of clustering results. 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.

Generally, cluster validity has three aspects. First, the quality of clusters can be measured in terms of *homogeneity* and *separation* on the basis of the definition of a cluster: objects within one cluster are similar to each other, while objects in different clusters are dissimilar with each other. The second aspect relies on a given “ground truth” of the clusters. The “ground truth” could come from domain knowledge, such as known function families of
Cluster validation is based on the agreement between clustering results and the “ground truth”. The third aspect of cluster validity focuses on the reliability of the clusters, or the likelihood that the cluster structure is not formed by chance. In this section, we will discuss these three aspects of cluster validation.

There are various definitions for the homogeneity of clusters which measures the similarity of data objects in cluster $C$. For example, $H_1(C) = \frac{\sum_{O_i, O_j \in C, O_i \neq O_j} \text{Similarity}(O_i, O_j)}{|C|(|C|-1)}$. This definition represents the homogeneity of cluster $C$ by the average pairwise object similarity within $C$. An alternate definition evaluates the homogeneity with respect to the “centroid” of the cluster $C$, i.e., $H_2(C) = \frac{1}{|C|} \sum_{O_i \in C} \text{Similarity}(O_i, \bar{O})$, where $\bar{O}$ is the “centroid” of $C$. Other definitions, such as the representation of cluster homogeneity via maximum or minimum pairwise or centroid-based similarity within $C$ can also be useful and perform well under certain conditions. Cluster separation is analogously defined from various perspectives to measure the dissimilarity between two clusters $C_1, C_2$. For example, $S_1(C_1, C_2) = \frac{\sum_{O_i \in C_1, O_j \in C_2} \text{Similarity}(O_i, O_j)}{|C_1||C_2|}$ and $S_2(C_1, C_2) = \text{Similarity}(\bar{O}_1, \bar{O}_2)$.

Since these definitions of homogeneity and separation are based on the similarity between objects, the quality of $C$ increases with higher homogeneity values within $C$ and lower separation values between $C$ and other clusters. Once we have defined the homogeneity of a cluster and the separation between a pair of clusters, for a given clustering result $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$, we can define the homogeneity and the separation of
CHAPTER 2. RELATED WORK

C. For example, Sharan et al. [75] used definitions of $H_{ave} = \frac{1}{N} \sum_{C_i \in \mathcal{C}} ||C_i|| \cdot H_2(C_i)$ and $S_{ave} = \frac{1}{\sum_{C_i \neq C_j} ||C_i|| \cdot ||C_j||} \sum_{C_i \neq C_j} (||C_i|| \cdot ||C_j||) S_2(C_i, C_j)$ to measure the average homogeneity and separation for the set of clustering results $\mathcal{C}$.

If the “ground truth” of the cluster structure of the data set is available, we can test the performance of a clustering process by comparing the clustering results with the “ground truth”. Given the clustering results $\mathcal{C} = \{C_1, ..., C_p\}$, we can construct a $n \times n$ binary matrix $\mathcal{C}$, where $n$ is the number of data objects, $C_{ij} = 1$ if $O_i$ and $O_j$ belong to the same cluster, and $C_{ij} = 0$ otherwise. Similarly, we can build the binary matrix $\mathcal{P}$ for the “ground truth” $\mathcal{P} = \{P_1, ..., P_s\}$. The agreement between $\mathcal{C}$ and $\mathcal{P}$ can be disclosed via the following values:

- $n_{11}$ is the number of object pairs $(O_i, O_j)$, where $C_{ij} = 1$ and $P_{ij} = 1$;
- $n_{10}$ is the number of object pairs $(O_i, O_j)$, where $C_{ij} = 1$ and $P_{ij} = 0$;
- $n_{01}$ is the number of object pairs $(O_i, O_j)$, where $C_{ij} = 0$ and $P_{ij} = 1$;
- $n_{00}$ is the number of object pairs $(O_i, O_j)$, where $C_{ij} = 0$ and $P_{ij} = 0$.

Some commonly used indices [37] have been defined to measure the degree of similarity between $\mathcal{C}$ and $\mathcal{P}$:

(2.2.1) Rand index: $Rand = \frac{n_{11} + n_{00}}{n_{11} + n_{10} + n_{01} + n_{00}}$;

(2.2.2) Jaccard coefficient: $JC = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$. 
(2.2.3) Minkowski measure: \( Minkowski = \sqrt{\frac{n_{10} + n_{01}}{n_{11} + n_{01}}} \).

The \textit{Rand index} and the \textit{Jaccard coefficient} measure the extent of agreement between \( \mathcal{C} \) and \( \mathcal{P} \), while \textit{Minkowski measure} illustrates the proportion of disagreements to the total number of object pairs \((O_i, O_j)\), where \( O_i, O_j \) belong to the same set in \( \mathcal{P} \). It should be noted that the \textit{Jaccard coefficient} and the \textit{Minkowski measure} do not (directly) involve the term \( n_{00} \). These two indices may be more effective in gene-based clustering because a majority of pairs of objects tend to be in separate clusters and the term \( n_{00} \) would dominate the other three terms in both good and bad solutions. Other methods are also available to measure the correlation between the clustering results and the “ground truth” [37]. Again, the optimal index selection is application-dependent.

### 2.3 Outlier Detection

Outlier detection is concerned with discovering the exceptional behaviors of certain objects. It is an important branch in the field of data mining with numerous applications, including credit card fraud detection, discovery of criminal activities, discovery of computer intrusion, and etc. In some sense it is at least as significant as cluster detection.
There are numerous studies on outlier detection. D. Yu etc. [91] proposed an outlier detection approach termed FindOut as a by-product of WaveCluster [76] which removes the clusters from the original data and thus identifies the outliers by applying signal-processing techniques. E. M. Knorr etc. [54] detected a distance-based outlier which is a data point with a certain percentage of the objects in the data set having a distance of more than $d_{\text{min}}$ away from it. S. Ramaswamy etc. [68] further extended it based on the distance of a data point from its $k^{th}$ nearest neighbor and identified the top $n$ points with largest $k^{th}$ nearest neighbor distances as outliers. M. M. Breunig etc. [19] introduced the concept of local outlier and defined local outlier factor (LOF) of a data point as a degree of how isolated the data point is with respect to the surrounding neighborhood. Aggarwal etc. [5] considered the problem of outlier detection in subspace to overcome dimensionality curse.

### 2.4 Indexing Multi-Dimensional Data

Recently large volumes of data with high dimensionality are being generated in many fields. Many approaches have been proposed to index high-dimensional data sets for efficient querying. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data sets efficiently since it may greatly increase the amount of data accessed for a query.
Among those approaches, the ClusterTree [92] is the first work towards building efficient index structure from clustering for high-dimensional data sets. The ClusterTree approach builds an index structure on the cluster structures to facilitate efficient queries.

Existing multidimensional tree-like indexing approaches can be further classified into two categories: space partitioning and data partitioning.

A data partitioning approach partitions a data set and builds a hierarchy consisting of bounding regions. The bounding regions at higher levels contain the lower regions and the bounding regions at the bottom level contain the actual data points. Popularly used index structures include R-Tree, R*-tree, SS-Tree, SS+-Tree and SR-Tree. An R-tree [36] is a height-balanced tree with index records in its nodes. There are two kinds of nodes: internal and leaf nodes. All the nodes have minimum bounding rectangles (MBR) as page region. R-tree is used to store rectangular regions of an image or a map. R-trees are very useful in storing very large amounts of data on disk. One disadvantage of R-trees is that the bounding boxes (rectangles) associated with different nodes may overlap. The R*-tree [11] is an R-tree variant which incorporates a combined optimization of area, margin and overlap of each enclosing rectangle in the tree nodes. It reduces the overlapping between the MBRs of neighboring nodes, reduces the volume of the MBRs and improves the storage utilization.
In contrast to the R-Tree, SS-Tree [1] uses hyper-spheres as region units. Each hyper-
sphere is represented by a centroid point, and a radius large enough to contain all of the
underlying data points. Queries on the SS-Tree are very efficient because it only needs to
calculate similarity between a region and the query point. The SS$^+$-Tree [56] is a variant
of the SS-Tree with a modified splitting heuristic to optimize the bounding shape for each
node.

SR-Tree [51] is an index structure which combines the bounding spheres and rectangles
for the shapes of node regions to reduce the blank area. The region for a node in it is
represented by the intersection of a bounding sphere and rectangle so that the overlapping
area between two sibling nodes is reduced. It takes the advantages of both rectangles and
spheres. However, the storage required for the SR-Tree is larger than the SS-Tree because
the nodes in the SR-Tree need to store the bounding rectangles and bounding spheres.
Consequently, the SR-Tree requires more CPU time and more disk accesses than the SS-
Tree for insertions.

Space partitioning approaches recursively divide a data space into disjoint subspaces. K-
D-B Tree and Pyramid-Tree are this type. The K-D-B Tree [69] partitions a $d$-dimensional
data space into disjoint subspaces by $(d - 1)$-dimensional hyper-planes which are alternate-
ly perpendicular to one of the dimension axes. Due to the disjointness among its nodes
at the same level, the K-D-B Tree has the advantage that the search path for querying a data
point is unique. The Pyramid-Tree [13]’s main idea is to divide the data space first into 2d pyramids, each sharing the center point as its peak (the tip point of a pyramid). Each pyramid is then sliced into slices parallel to the base of the pyramid, and each slice forms a data page. The range query under this index structure can be efficiently processed for both low- and high-dimensional data sets. However, the partitioning of the Pyramid-Tree can not ensure that the data points in one data page are always neighbors.

Among those approaches, the ClusterTree is the first work towards building efficient index structures from clustering for high-dimensional data sets. The ClusterTree has the advantage of supporting efficient data query for high-dimensional data sets. A ClusterTree is a hierarchical representation of the clusters of a data set. It organizes the data based on their cluster information from coarse level to fine, providing an efficient index structure on the data according to clustering. Like many other structures, it has two kinds of nodes: internal and leaf nodes. The internal nodes include pointers to subclusters of this cluster, the bounding sphere for the subclusters and the number of data points in the subclusters. The leaf nodes contain pointers to the data points. For each cluster, the ClusterTree calculates the following parameters: the number of data points, the centroid \( \mathbf{c} \), and the volume of the minimum bounding sphere \( S \). The centroid \( \mathbf{c} = \langle c_1, c_2, \cdots, c_d \rangle \) can be calculated by:

\[
(2.4.1) \quad c_i = \frac{\sum_{j=1}^{N} o_{ji}}{N}, \quad 1 \leq i \leq d,
\]

where \( N \) is the number of the data points in the cluster and \( o_{ji} \) is the \( i \)-th value of data
point $o_j$ in the cluster. Thus, each cluster is represented by a hyper-sphere $S$. So there may be some empty regions which contain no data, and two bounding hyper-spheres of two different clusters may overlap. We can define the density of the cluster as:

$$\text{Density}_c = \frac{\text{number of points in } C}{\text{volume of } S} = \frac{\text{number of points in } C}{\frac{2\pi^{d/2}r^d}{d\Gamma\left(\frac{d}{2}\right)}}.$$  

(2.4.2) 

The gamma function $\Gamma(x)$ is defined as:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt,$$

(2.4.3) 

where $\Gamma(x + 1) = x\Gamma(x)$ and $\Gamma(1) = 1$. When the density of a cluster falls below a pre-selected threshold or the number of the data points in the cluster is larger than a pre-selected threshold, the cluster will be decomposed into several smaller clusters (subclusters).
Chapter 3

A Shrinking-Based Clustering Algorithm

Existing data analysis techniques have difficulty in handling multi-dimensional data. Multi-dimensional data has been a challenge for data analysis because of the inherent sparsity of the points. In this chapter, we first present a novel data preprocessing technique called shrinking which optimizes the inherent characteristic of distribution of data. 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 (and evaluated by their compactness). 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.

3.1 Introduction

The generation of multi-dimensional data has proceeded at an explosive rate in many disciplines with the advance of modern technology. Data preprocessing procedures can greatly benefit the utilization and exploration of real data. Clustering is useful for discovering groups and identifying interesting distributions in the underlying data. In this chapter, we first present a data shrinking technique for preprocessing; then, we propose a cluster detection approach by finding the connected components of dense cells and a cluster evaluation approach based on the compactness of clusters.
Data preprocessing is commonly used as a preliminary data mining practice. It 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: data cleaning, data integration, data transformation and data reduction. Data cleaning can be applied to remove noise and correct inconsistencies in the data. Data integration merges data from multiple sources into a coherent data store. Data transformation may improve the accuracy and efficiency of mining algorithms involving distance measurements. Data reduction can reduce the data size. These data processing techniques, when applied prior to mining, can substantially improve the overall quality of the patterns mined and/or the time required for the actual mining[47].

Cluster analysis is used to identify homogeneous and well-separated groups of objects in databases. The need to cluster large quantities of multi-dimensional data is widely recognized. It is a classical problem in the database, artificial intelligence, and theoretical literature, and plays an important role in many fields of business and science.

Each of the existing clustering algorithms has both advantages and disadvantages. The most common problem is rapid degeneration of performance with increasing dimensions [42], particularly with approaches originally designed for low-dimensional data. The difficulty of high-dimensional clustering is primarily due to the following characteristics of
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. For example, the algorithms in [64, 93] can efficiently find convex or spherical clusters, but they fail to detect non-spherical clusters because of their specific definition of similarity criteria.

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 3.1.1. 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.

There are also other algorithms related to data movement[82, 23], however, the existing ones are not suitable for the high dimensional data of large size because based on their
Figure 3.1.1: 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.

definition, the time to run their process is $O(n^2 \times p)$ where $n$ is the size of the input data and $p$ is the number of iterations in the iterative process. So they are very time-consuming.

A lot of approaches [21, 59, 60, 61] have been proposed for evaluating the results of a clustering algorithm. 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.

3.1.1 Proposed approach

In this chapter, we first present a novel data preprocessing technique named shrinking which optimizes the inherent characteristic of distribution of data. For a real data set, the natural data groups (if existing) it contains may be very sparse. In the data preprocessing
step, if we could make data points move towards the centroid of the data groups they belong to, the natural sparse data groups will become denser, thus easier to be detected, and noises can be further isolated. Intuitively, a dense area “attracts” objects in sparse areas surrounding it, and becomes denser. Here we assume a data point is attracted to neighboring data points, and it moves towards the way the attraction is the strongest. In other words, the direction it is attracted to is determined by the distribution of its neighboring data points. Our data shrinking preprocessing computes a simulated movement of each data point in a data set that reflects its “attraction” to neighboring data points. We can also refer to the concept of infiltration mechanism[85] in which materials such as water move from denser areas to sparser ones whereas in our case, the data point will move to a denser area nearby. When computing the attraction on a data point, those points far away from this data point can be ignored due to the little effect they impose on it. 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 categories.

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, 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 includes 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 3.1.1. 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.

Although the idea of moving data points according to the density gradient has been around quite some time [48], this chapter distinguishes mainly in the following two aspects:

- A grid-based shrinking and evaluation approach is proposed. Instead of choosing a grid with a fixed cell size, we use a sequence of increasing grid sizes to catch the cluster structures of the input data in different scales. At each scale, two grids are used, the second one is shifted diagonally from the first one.
CHAPTER 3. A SHRINKING-BASED CLUSTERING ALGORITHM

• We integrate a compactness-based cluster evaluation into the framework. The compactness-based evaluation computes both inter-cluster and intra-cluster distances (which model the inter-cluster and intra-cluster relationships, respectively) and evaluate a cluster’s compactness with the ratio of the second distance to the first one.

The remainder of this paper is organized as follows. Section 3.2 introduces methods for the selection of multiscale grids for use in data shrinking and cluster detection. Section 3.3 discusses the data-shrinking process. In Section 3.4, a simple grid-based cluster detection method is presented. In Section 3.5, we discuss our definition of compactness as pertains to evaluating and selecting clusters. Section 5.4 presents experimental results, and concluding remarks are offered in Section 7.1.

3.2 Choosing Grid Scales for the Shrinking-based Clustering Approach

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.
CHAPTER 3. A SHRINKING-BASED CLUSTERING ALGORITHM

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 are 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 $[0, 1)^d \subset \mathbb{R}^d$.

We apply a simple histogram-based approach to get reasonable grid scales for the data-shrinking process. We scan 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 a histogram denotes the number of data points in a certain segment on this histogram.

We set up a number $\beta$ as a quantity threshold. It is used in the following algorithm to help
generate *density spans*. Here we first give the definition of **density span** which will help understand the algorithm:

**Definition 1:** A **density span** is a combination of consecutive bins’ segments on a certain dimension in which the amount of data points exceeds $\beta$. A **size** of a density span is the sum of the sizes of the bins it includes.

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 and merge it with its neighboring bins until the total amount of data points in these bins exceeds $\beta$. At each step, we check the number of points in the bin on the left side and the one on the right side of the currently span, and choose the bin with more points in it to merge with. Thus a density span is generated as the combination of the segments of these bins. If a current span has less than $\beta$ data points, but its left and right neighbors have both been assigned to a precious span already, we stop the operation on the current span and call it as an *incomplete span* which will not be considered in the following procedure of generating multiple grid scales. The operation is continued until all the non-empty bins of this histogram is in some density spans or some incomplete spans. Each histogram has a set of density spans.

Figure 3.2.1 shows an example of this density span generation operation. Here we just demonstrate two density spans on this histogram. Bin 21 is the one with largest amount of
Figure 3.2.1: An example of density span acquirement

data points. We start from Bin 21, check its neighboring bins 20 and 22, and choose bin 20 to merge with because it has more points than bin 22. Next we check the neighboring bins of the combination of the segments of bin 20 and 21, which are bin 19 and bin 22, and choose bin 19 because it has more points than bin 22. In this way the neighboring bins are merged until the amount of data points included exceeds $\beta$. 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, first including bin 8, then including bin 6, 9, 10 and 5, step by step until the amount of data points included exceeds $\beta$.

After the density span generation operation mentioned above, we have a set $T$ of density spans with different sizes. We cluster the density spans by their sizes. We do the procedure
as follows. First we sort the sizes by ascending order. Then starting from the smallest size \( s_0 \), we include the current size into cluster \( T_1 \) until we come across a size \( s' \) such that \( s' - s_0 > s_0 \cdot 5\% \). Then we start from \( s' \) and do the same procedure to get cluster \( T_2 \), and so on. For each cluster \( T_i \), we denote the number of sizes in it as \( N_i \), and denote the average value of the sizes in it as \( S_i \). we sort \( S_i \) based on \( N_i \) by descending order and choose first \( K_s \) ones as the multiple scales for the following data shrinking and cluster detection procedures. In other words, those sizes of density spans which appear often are chosen. Algorithm 1 describes the procedure of the density span generation on a certain dimension.

The value \( \beta \) depends on the size of the input data set \( X \). Normally it can be set as a certain percentage of the number of data points in \( X \). There is a balance in choosing a value for \( K_s \): a smaller \( K_s \) can increase the precision of cluster detection, while a larger \( K_s \) 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 \( O(B_n \log B_n) \).
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 $\beta$;
3) Repeat step 2 until all non-empty bins are included in some density spans or some incomplete spans;
4) Output the density span set.

The multiscale gridding scheme proposed above 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 3.2.2 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.
3.3 Data Shrinking

We start with the first step of the proposed method: data shrinking. In this step, 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.

3.3.1 Space subdivision

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

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

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 $k = 1,2,...,d$. The neighboring cells of a cell $C$ are also called the surrounding cells of $C$. This arrangement is shown in Figure 3.3.1.

For each data point $\vec{X}_i$, the cell containing the point can be easily found; this cell is denoted
as $Cell(\hat{X}_i)$. 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 4.2.3, 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

\begin{equation}
DenseCellSet = \{C_1, C_2, \ldots, C_m\}.
\end{equation}

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

\begin{equation}
DataCentroid(C) = \frac{\sum_{j=1}^{k} \hat{X}_{ij}}{k},
\end{equation}

where $\{\hat{X}_{ij}\}_{j=1}^{k}$ 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 $O(n)$. 

Figure 3.3.1: Surrounding cells (in gray) of the cell $C$ (in black)
In high-dimensional spaces, ignoring the sparse cells can be problematical. Figure 3.3.2(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 3.3.2(a) and (b).

![Diagram of grid points](image)

Figure 3.3.2: (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.

The idea of using several shifted grids to improve the histograms accuracy was first published in 1985 by D.W. Scott [26]. When handling data of large size, effectiveness and efficiency are always the two main goals we need to achieve and balance. Regarding the shifted grids, \( 2^d \) shifted grids are needed to ensure that we get a dense cell for each clustered point set. However, that’s practically infeasible.

We choose two grids at each scale with the second one shifted diagonally from the first one because it will improve the efficiency of our algorithm. Also the diagonally shifted
grid best complements the original one (as compared with each individual other), though
the complement is not complete. Realizing the problem of the incompleteness, we used a
sequence of grid sizes.

In [65] a random selection of shifts has been used. However, to balance the effectiveness
and efficiency, this approach has to make a decision on how many shifts will be used. Our
approach attempts to handle this problem by changing grid sizes, which has two advan-
tages: (1) It can also be used for catching the cluster structures in different scales. (2)
When choosing a sequence of grid sizes, we used the approach of density span genera-
tion (Algorithm 1). As a comparison, the random-selection-of-shift approach has to choose
among $2^d$ possible shifts to find the properly shifted grids.

### 3.3.2 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. 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 3.3.3(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 3.3.3(a) becomes the isolated points shown in Figure 3.3.3(b).

![Figure 3.3.3: 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.](image)

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 $i$th iteration becomes

$$\{\bar{X}_1^i, \bar{X}_2^i, \ldots, \bar{X}_n^i\}.$$
and the set of dense cells is

\[ \text{DenseCellSet}^i = \{ C_1^i, C_2^i, ..., C_m^i \}. \]

Respectively, we assume that the dense cells have

\[ n_1, n_2, ..., n_m, \]

points and their data centroids are

\[ \Phi_1, \Phi_2, ..., \Phi_m. \]

For a dense cell \( C_j^i \), we suppose that its surrounding dense cells are \( C_{jk}^i \), for \( k = 1, 2, ..., w \).

Then the data centroid of these surrounding cells is

\[ \sum_{k=1}^{w} n_{jk} \times \Phi_{jk} \]

\[ \sum_{k=1}^{w} n_{jk}, \]

which is denoted as \( \Phi_j^s \). The movement for cell \( C_j^i \) in the \( i \)th iteration is

\[ \text{Movement}(C_j^i) = \begin{cases} 
\Phi_j^s - \Phi_j \text{ if } \| \Phi_j^s - \Phi_j \| \geq T_{mv} \times \frac{1}{K} \\
\text{and } \sum_{k=1}^{w} n_{jk} > n_j; \\
0 \text{ otherwise}, 
\end{cases} \]

where \( \| \Phi_j^s - \Phi_j \| \) is the distance between the two centroids \( \Phi_j^s \) and \( \Phi_j \), \( T_{mv} \) is a threshold to ensure that the movement is not too small, and \( \frac{1}{K} \) is the side length of grid cells, as discussed in the last subsection of this section. \( T_{mv} \) is usually a number between 0.5 and \( 0.5 \times \sqrt{d} \). Formula 3.3.6 states that, if the distance between \( \Phi_j^s \) and \( \Phi_j \) 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 $\bar{X}$ in cell $C_j^i$ is moved to $\bar{X} + \Phi_j^i - \Phi_j$.

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 $i$th iteration, the time used to move all points is $O(m^2 + n)$. Thus, the time required for the $i$th iteration is $O(m^2 + n \log n)$, where $O(n \log n)$ time is used for the subdivision of space.

### 3.3.3 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 3.3.4. 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.
The average movement of all points in each iteration is checked to determine the stability of the data set. Suppose that in the $i$th iteration, the movements for the $n$ points are $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$, respectively. Then the average movement is $\frac{\sum_{j=1}^{n} \|\vec{v}_j\|}{n}$. If the average movements for two consecutive iterations are both less than $T_{\text{amv}}$, where $T_{\text{amv}}$ is a threshold, then the data set is considered stabilized and the shrinking process is terminated.

Due to the complicated high-dimensional structures in most real data, it would be a very challenging problem, if solvable, to give a theoretical analysis of the number of iterations needed for the real data to get stabilized. To control the total computation time, we set an upper bound, $T_{\text{it}}$, for the number of iterations. If the number of iterations exceeds $T_{\text{it}}$ before the data is stabilized, then the shrinking process is also terminated.

### 3.3.4 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 $T_{dn1} \times (\frac{1}{K})^d$. Thus the number of dense cells must not exceed $\frac{n}{T_{dn1} \times (\frac{1}{K})^d}$. Given a desired value $\tilde{M}$, we can choose a value $T_{dn1}$ such that $\frac{n}{T_{dn1} \times (\frac{1}{K})^d} \leq \tilde{M}$, thus ensuring 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 $\frac{1}{K}$, most non-empty cells will be very sparse, containing only one or two points each. In such instances, the side length $\frac{1}{K}$ is too small and a larger-scale grid should be used. This is due to the fatal weakness of grid-based approach for high-dimensional data analysis. As we discussed previously in this section, unless $2^d$ shifted grids are used, it is not guaranteed to have a dense cell for each clustered point set, which makes such a case possible: while the input data set has clusters, it appears sparsely in all cells of the chosen grids. A possible solution to this problem is to
combine the grid-based approach with a nearest-neighbor search and use the result from a
earest-neighbor search to choose proper grids. Due to the limited pages of this paper, we
cannot investigate this in more details.

3.4 Cluster detection

The second step of our method is cluster detection. Since the data-shrinking process gen-
erates 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 $\frac{1}{k}$, after the data-shrinking process is performed on the input
data set, we find the dense cells. Neighboring dense cells are connected and a neighborhood
graph of the dense cells is constructed. Each connected component of the neighborhood
graph is a cluster.

The cluster-detection method is conducted on two interleaved grids. This avoids the prob-
lem caused by points clustered near a vertex of a grid and separated in different cells,
as discussed in Subsection 3.3.1. 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
CHAPTER 3. A SHRINKING-BASED CLUSTERING ALGORITHM

$C_1 \in DC_1$ and $C_2 \in DC_2$ are called neighbors if $C_1 \cap C_2 \neq \emptyset$. The neighborhood graph of dense cells, $G$, is a pair $G = \langle DC_1 \cup DC_2, E \rangle$, where $E$ is the set of neighboring pairs in $DC_1 \cup DC_2$. The edge set $E$ can be represented by a matrix. Let $|DC_1| = m_1$, $|DC_2| = m_2$, and $DC_1 \cup DC_2 = \{C_i\}_{i=1}^{m_1+m_2}$. Then $E = (E_{ij})_{(m_1+m_2) \times (m_1+m_2)}$, where

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

(3.4.1)

We then run a breadth-first search algorithm (see pages 469-472 in [24]) to find the components of graph $G$.

The time and space required for the breadth-first search algorithm are both $O(|DC_1 \cup DC_2| + |E|)$. 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.
3.5 Cluster evaluation and selection

After the cluster detection step, we evaluate the clustering results. There are several ways to define what is a good clustering ([53].etc). Most conventional clustering validity measurements [21, 59, 60, 61] 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 [21, 59], 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.
3.5.1 Compactness graphing

We first define compactness in a weighted graph. Then we give an algorithm to compute compact subsets and prove that the compactness defined in a graph is equivalent to that defined in its minimal spanning tree. Thus the problem of finding compact subsets (i.e. clusters) in a graph is reduced to compute its minimal spanning tree. Let $G = \langle V, E \rangle$ be a fixed graph, where $V$ is the vertex set and $E$ is the edge set. Let $w : E \rightarrow \mathbb{R}^+$ 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 any graph $G = \langle V, E \rangle$ with positive weight function $w$, and any connected\(^1\) subset $T \subseteq V$, let $L = \{w(p, q) \mid (p, q) \in E, p \in T, q \not\in T\}$. We define the external connecting distance of $T$ with respect to $G$ as: $ECD(T; G, w) = \min L$. If $L$ is empty, then define $ECD(T; G, w)$ to be infinite.

In the above definition, ”min L” denotes the minimum number in set L.

**Definition 3:** For any graph $G = \langle V, E \rangle$ with positive weight function $w$, and any connected subset $T \subseteq V$, let $L = \{l \in \mathbb{R}^+ \mid X\}$, where $X = \langle T, \{(p, q) \in E \mid p \in T, q \in T \wedge w(p, q) \leq 1\} \rangle$ is connected.

\(^1\) $T$ is connected means subgraph $\langle T, \{(p, q) \in E \mid p, q \in T\} \rangle$ is connected.
is a connected subgraph of $G$. We define the **internal connecting distance** of $T$ with respect to $G$ as: $ICD(T; G, w) = \inf L$.

In the above definition, "\(\inf L\)" denotes greatest lower bound of set $L$.

$ECD(T; G, w)$ gives the length of a shortest edge connecting $T$ and $V - T$. $ICD(T; G, w)$ gives the shortest length to “maintain” $T$ connected. It is easy to see that $ECD(T; G, w)$ is infinite iff $T$ is a connected component of $G$, and for any connected subset $T$, $ICD(T; G, w)$ is finite, and there is a connected subgraph $<T, H>$ such that all edges in $H$ have weights no more than $ICD(T; G, w)$.

**Definition 4:** For any graph $G = <V, E>$ with positive weight function $w$, and any connected subset $T \subseteq V$, we define $T$’s **compactness** as $Compactness(T; G, w) = \frac{ECD(T; G, w)}{ICD(T; G, w)}$.

$T$ is **compact** (or $T$ is a **cluster**) if its compactness is greater than 1, i.e., $ECD(T; G, w) > ICD(T; G, w)$. We denote the set of all clusters in $G$ as $\text{ClusterSet}(G, w)$.

Algorithm 2 is an algorithm to calculate all clusters in a weighted graph. This algorithm works similarly to connected-component-finding algorithms. In the beginning, each vertex is a component. Then starting from the shortest edge, we merge these components with edges in the order of weight-ascending. Each edge connecting two components will merge the two components into one. In the merging process, output the components with compactness greater than one.
We have following theorem to guarantee that the above algorithm is correct.

**Theorem 1:** For any graph $G =< V, E >$ with positive weight function $w$, let $C$ be the output of ClusterFinding$(G, w)$. Then we have: $C = \text{ClusterSet}(G, w)$.

**Proof:** First we prove $C \subseteq \text{ClusterSet}(G, w)$, i.e., for any $P \in C$, $P \in \text{ClusterSet}(G, w)$.

If $P$ appears in the final $D$, i.e., $P$ is added to $C$ by the operation of line 14, then $P$ is a connected component of $G$, thus it is a cluster, i.e., $P \in \text{ClusterSet}(G, w)$. From now on, suppose $P \in C$ doesn’t appear in the final $D$. Then $P$ is added to $C$ by the operation of line 8 or 9. It is obvious that for such $P$, $\text{ecl}(P) < \infty, \text{icl}(P) < \infty$. To prove $P \in \text{ClusterSet}(G, w)$, it suffices to prove $\text{ecl}(P) = \text{ECD}(P; G, w)$ and $\text{icl}(P) = \text{ICD}(P; G, w)$.

To prove $\text{ecl}(P) \geq \text{ECD}(P; G, w)$, by the algorithm, there is some $(p, q) \in E$ such that $p \in P \in D, q \in Q \in D, P \neq Q,$ and $w(p, q) = \text{ecl}(P)$. It is obvious that at all time, $D$ is a subdivision of $V$. So $q \notin P$. By the definition of $\text{ECD}$, we have $\text{ecl}(P) \geq \text{ECD}(P; G, w)$.

Here we do not present the proof of $\text{ecl}(P) = \text{ECD}(P; G, w), \text{icl}(P) = \text{ICD}(P; G, w)$, and
we omit the proof of $\text{ClusterSet}(G, w) \subseteq C$. 

\begin{algorithm}
\caption{(ClusterFinding(G,w))}
\begin{itemize}
    \item[1)] Let $D = \{\{p\} | p \in V\}$;
    \item[2)] Let $ecl : D \rightarrow \mathbb{R}^+$ be a function;
    \item[3)] Let $icl : D \rightarrow \mathbb{R}^+$ be a function;
    \item[4)] Sort $E$ by the weights of all edges in ascending order;
    \item[5)] For each edge $(p,q) \in E$, if $p \in P \in D, q \in Q \in D$ and $P \neq Q$,
    \item[6)] let $ecl(P) = ecl(Q) = w(p,q)$;
    \item[7)] if $ecl(P) > icl(P)$, add $P$ to $C$;
    \item[8)] if $ecl(Q) > icl(Q)$, add $Q$ to $C$;
    \item[9)] remove $P, Q$ from $D$; Add $P \cup Q$ to $D$;
    \item[10)] let $icl(P \cup Q) = w(p,q)$;
    \item[11)] For each $X \in D$,
    \item[12)] let $ecl(X) = \infty$;
    \item[13)] add $X$ to $C$; //Connected components are trivial clusters
    \item[14)] Output $C$;
\end{itemize}
\end{algorithm}

\textbf{Corollary 1:} For any graph $G = < V, E >$ with positive weight function $w$, and any $P \in$
ClusterSet$(G, w)$, we have $ECD(P; G, w) = ecl(P)$ and $ICD(P; G, w) = icl(P)$, where $ecl, icl$ are functions defined in Algorithm 2.

If we compare Algorithm 2 and Kruskal’s algorithm for computing minimal spanning tree (See page 505 of [24]), we can find that for each edge $e = (p, q) \in E$, if it is not in the minimal spanning tree given by Kruskal’s algorithm (i.e., when $(p, q)$ is processed, $p, q$ are in the same component in $D$), then $e$ has no effect to the output of Algorithm 2. Then we get following corollary.

**Corollary 2:** For any graph $G = < V, E >$ with positive weight function $w$, let $SG = < V, SE >$ be a minimal spanning tree of $G$, and $sw = w|_{SE} : SE \to \mathbb{R}^+$. Then we have
(1) $\text{ClusterSet}(G, w) = \text{ClusterSet}(SG, sw)$; and
(2) $\forall P \in \text{ClusterSet}(G, w), \text{Compactness}(P; G, w) = \text{Compactness}(P; SG, sw)$.

For any finite point set $V$ in Euclidean space $\mathbb{R}^K$, a minimal spanning tree of its complete graph is always a subgraph of its Delaunay graph. So we have following result.

**Corollary 3:** For any finite point set $V$ in Euclidean space $\mathbb{R}^K$, let $G = (V, E)$ be its complete graph, and $DG = (V, DE)$ be its Delaunay neighborhood graph. For any $p, q \in \mathbb{R}^K$, let $d_E(p, q)$ be the Euclidean distance between $p, q$. Then we have
(1) $\text{ClusterSet}(G, d_E) = \text{ClusterSet}(DG, d_E)$; and
(2) $\forall P \in \text{ClusterSet}(G, d_E), \text{Compactness}(P; G, d_E) = \text{Compactness}(P; DG, d_E)$. 
From Algorithm 2, we can easily see that any two clusters don’t intersect if one doesn’t include the other.

**Corollary 4:** For any graph \( G = \langle V, E \rangle \) with positive weight function \( w \), and any \( P, Q \in \text{ClusterSet}(G, w) \), we have \( P \subseteq Q \), \( Q \subseteq P \), or \( P \cap Q = \emptyset \). Thus if \( G \) is a connected graph, then under including relationship, all clusters form a tree, and \( V \) as a cluster is the root of the cluster tree.

Now we redefine *internal connecting distance (ICD)*, *external connecting distance (ECD)* and *compactness* based on minimum spanning tree.

**Definition 5:** For a connected subset \( S \) of \( V \), let \( \text{MST}(S) \) be a minimum spanning tree of the minimal subgraph containing \( S \). The *internal connecting distance (ICD)* of \( S \), denoted as \( \text{ICD}(S; G, w) \), is defined as the length of a longest edge of \( \text{MST}(S) \). The *external connecting distance (ECD)* of \( S \), denoted as \( \text{ECD}(S; G, w) \), is defined as the length of a shortest edge connecting \( S \) and \( V - S \). The *compactness* of \( S \), denoted as \( \text{Compactness}(S; G, w) \), is defined as

\[
\text{Compactness}(S; G, w) = \frac{\text{ECD}(S; G, w)}{\text{ICD}(S; G, w)}.
\]

(3.5.1)

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

To evaluate a data set in a low-dimensional Euclidean space, we first construct its Delaunay graph \([7]\). 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.

### 3.5.2 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 3.5.1 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 3.5.1: 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 3.4, clusters are detected as the connected components of the
neighborhood 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 $DG = \langle DC, E \rangle$, where $E$ is the set of pairs of cells in $DC$. The weight 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 [24]). 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 $O(|DC|^2)$. 
3.5.3 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 3.2. 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 clustering result, a cluster can be a subset of another cluster. For example, the clusters of the data set in Figure 3.5.2(a) form a tree in Figure 3.5.2(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 $C_1 \cap C_2 = \emptyset$. 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 $S_1 \cap S_2 = \emptyset$. In situations where an inclusive relationship exists, all compact vertex sets form a tree.

Kleinberg [53] pointed out that there is no partitioning clustering algorithm satisfying all three of simple properties: essentially scale-invariance, a richness requirement that all
partitions be achievable, and a consistency condition on the shrinking and stretching of individual distances. The problem is due to the fact that the relationships among clusters of a data set may not have a flat structure. In general, they have a hierarchical structure. We proposed a grid-based multi-scale (hierarchical) clustering algorithm. With our compactness approach, the clusters are organized as a tree, as mentioned above. Kleinberg’s result illustrates the necessities of cluster hierarchies for data clustering to some extent.

3.6 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. Next we give our experimental results on high-dimensional data sets to show the scalability of our approach. 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 alone 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. \( T_{it} \) is the upper bound set for the number of iterations to control the total computation time. Some data sets which has natural clusters will converge to form denser clusters under proper grid scales, others which either do not have natural clusters at all, or are not under proper grid scales, do not converge at all. In our experiments, we empirically found that after at most 10 iterations, the movements of most of the data sets which converge eventually will be stabilized, so we set \( T_{it} \) as 10.

Other parameters are optimized as follows. For each parameter, we did experiments to check its values which are best for the runtime of the algorithm or the performance in terms of the resulted clusters. Values which are best for runtime usually are not best for performance though. So we shall make tradeoff between these two best values to get the value of this parameter.

Due to the space limitation, here we just briefly present the procedure of determining the value of \( T_{amv} \), the threshold for average movement. We did experiments with \( T_{amv} \) of various values (e.g., 0.04 means \( T_{amv} \) is 4\% of the data range), on different data sets. Figure 5.4.1 shows the performance (the average of precisions and recalls of the resulting clusters) and runtime of our algorithm on Wine data. From the figure we can see that \( T_{amv} \) values best for runtime maybe not be best for performance. We made tradeoff between these two best values on different data sets, and set the value of \( T_{amv} \) as 2.4\% of the data range.
CHAPTER 3. A SHRINKING-BASED CLUSTERING ALGORITHM

3.6.1 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 3.6.1: The runtime and performance of Shrinking on wine data with different average movement threshold $T_{amv}$.

Figure 3.6.2: Shrinking process on the data set $DS_1$ with cell size $0.1 \times 0.1$. (a) 2-dimensional data set $DS_1$, (b) the data set after the shrinking process
3.6.2 Experiments on high-dimensional datasets

To test the scalability of our algorithm over dimensionality and data size, we designed a synthetic data generator to produce data sets with clusters. The sizes of the data sets vary from 5,000, 10,000, ... to 60,000, and the dimensions of the data sets vary from 10, 20, ... to 60. Each data set contains five clusters, with the points in each cluster generated in normal distributions. We set different values like 200, 350, 500, 650, and 800 to the centers of different clusters on each dimension (we can regard the center of a synthetic clusters as the mean of the data distribution in this cluster), and set the standard deviation $\sigma$ for different clusters on each dimension as 10, 5, 7, 9 and 4, respectively. An additional 5% of data points are added randomly to each data set as noise. Clusters were effectively detected by our algorithm in all tests performed on these high-dimensional data sets.

In the experiments, $\beta$ was set according to the size of the testing data set. Since for different synthetic data sets, the condition of convergence of the algorithm and the iteration numbers would be different, to make the comparison between data sets of different sizes and dimensionality fair, we fixed the number of iterations ranged as 5. Figure 5.4.2 shows the running time of 12 groups of data sets with dimensions increasing from 10 to 60. Each group has a fixed data size (the noisy data points are not counted). Figure 5.4.3 shows the running time of 6 groups of data sets with sizes increasing from 5,000 to 60,000. Each of these 6 groups has fixed dimensions.
3.6.3 Experiments on real data

In this subsection, 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 \textit{precision} and \textit{recall}.

For a detected cluster $C^s_i$ and a real cluster $C^r_i$, we define the precision of $C^s_i$ with respect to $C^r_i$ as $\frac{|C^s_i \cap C^r_i|}{|C^s_i|}$ and the recall as $\frac{|C^s_i \cap C^r_i|}{|C^r_i|}$. $C^s_i$ is called a corresponding cluster of $C^r_i$ if the
precision and recall of \( C_i^s \) with respect to \( C_i^o \) are higher than the precision and recall of \( C_i^s \) with respect to other real clusters.

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, -\( \alpha \) 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 [35].

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)[8], 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[93] 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 [93].

**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.

**Data sets and clustering results**

The three data sets were obtained from UCI Machine Learning Repository [10]. 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 5.4.4 shows the clustering results of our algorithm.
We applied CURE algorithm on the Wine Recognition data set, setting parameter values to different values. We set the cluster number parameter $k$ to 3 based on the ground truth of the Wine Recognition data set, set the shrinking factor $\alpha$ 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 $(\alpha, r)$ pair

Table 3.6.1: Clustering results of our algorithm for Wine data

<table>
<thead>
<tr>
<th></th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C^o_i</td>
<td>$</td>
<td>59</td>
</tr>
<tr>
<td>$</td>
<td>C^s_i</td>
<td>$</td>
<td>53</td>
</tr>
<tr>
<td>$</td>
<td>C^s_i \cap C^o_i</td>
<td>$</td>
<td>53</td>
</tr>
<tr>
<td>precision(%)</td>
<td>100</td>
<td>98.08</td>
<td>93.48</td>
</tr>
<tr>
<td>recall(%)</td>
<td>89.83</td>
<td>71.83</td>
<td>89.58</td>
</tr>
</tbody>
</table>

Table 3.6.2: Clustering result of CURE for Wine data as $\alpha=0.3$ and $r=30$

<table>
<thead>
<tr>
<th></th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C^o_i</td>
<td>$</td>
<td>59</td>
</tr>
<tr>
<td>$</td>
<td>C^s_i</td>
<td>$</td>
<td>72</td>
</tr>
<tr>
<td>$</td>
<td>C^s_i \cap C^o_i</td>
<td>$</td>
<td>54</td>
</tr>
<tr>
<td>precision(%)</td>
<td>75.00</td>
<td>82.00</td>
<td>56.52</td>
</tr>
<tr>
<td>recall(%)</td>
<td>91.52</td>
<td>57.77</td>
<td>54.16</td>
</tr>
</tbody>
</table>

We applied CURE algorithm on the Wine Recognition data set, setting parameter values to different values. We set the cluster number parameter $k$ to 3 based on the ground truth of the Wine Recognition data set, set the shrinking factor $\alpha$ 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 $(\alpha, r)$ pair

![Figure 3.6.5: 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.](image-url)
is (0.3, 30), (1, 20), (0.9, 10) or (0.5, 40). Here we just present one of the best results of CURE. Table 3.6.2 shows the clustering results of CURE algorithm when $\alpha$ is equal to 0.3, and $r$ is 30.

Figure 3.6.5(a) shows the cluster-ordering of OPTICS for the Wine data. The figure shows that there are roughly 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 5.4.4 and table 3.6.2.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$C^o_i$</th>
<th>$C^*_i$</th>
<th>$C^*_i \cap C^o_i$</th>
<th>precision(%)</th>
<th>recall(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>143</td>
<td>135</td>
<td>130</td>
<td>96.30</td>
<td>90.91</td>
</tr>
<tr>
<td>2</td>
<td>77</td>
<td>22</td>
<td>22</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>52</td>
<td>68</td>
<td>43</td>
<td>63.24</td>
<td>28.57</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>49</td>
<td>32</td>
<td>65.31</td>
<td>82.69</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>11</td>
<td>10</td>
<td>90.91</td>
<td>91.43</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3.6.3: Clustering result of our algorithm for Ecoli data

<table>
<thead>
<tr>
<th>$i$</th>
<th>$C^o_i$</th>
<th>$C^*_i$</th>
<th>$C^*_i \cap C^o_i$</th>
<th>precision(%)</th>
<th>recall(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>143</td>
<td>120</td>
<td>115</td>
<td>95.83</td>
<td>80.41</td>
</tr>
<tr>
<td>2</td>
<td>77</td>
<td>67</td>
<td>41</td>
<td>61.19</td>
<td>53.24</td>
</tr>
<tr>
<td>3</td>
<td>52</td>
<td>32</td>
<td>30</td>
<td>93.75</td>
<td>57.69</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>N/A</td>
<td>N/A</td>
<td>90.91</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>N/A</td>
<td>N/A</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>N/A</td>
<td>N/A</td>
<td>60.00</td>
<td>N/A</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3.6.4: Clustering result of CURE for Ecoli data as $\alpha=0.2$ and $r=30$

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 5.4.5 presents the clustering results. The real clusters $C^o_6$, $C^o_7$, and $C^o_8$ do not have corresponding clusters
CHAPTER 3. A SHRINKING-BASED CLUSTERING ALGORITHM

Table 3.6.5: 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.

<table>
<thead>
<tr>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
<th>i=8</th>
<th>i=9</th>
<th>i=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=1</td>
<td>j=2</td>
<td>j=3</td>
<td>j=4</td>
<td>j=5</td>
<td>j=6</td>
<td>j=7</td>
<td>j=8</td>
<td>j=8</td>
<td>j=8</td>
</tr>
<tr>
<td>$</td>
<td>C_o^i</td>
<td>$</td>
<td>1143</td>
<td>1144</td>
<td>1055</td>
<td>1056</td>
<td>1055</td>
<td>1055</td>
<td>1143</td>
</tr>
<tr>
<td>$</td>
<td>C_s^j</td>
<td>$</td>
<td>1098</td>
<td>1179</td>
<td>629</td>
<td>1051</td>
<td>480</td>
<td>833</td>
<td>2379</td>
</tr>
<tr>
<td>$</td>
<td>C_s^j \cap C_o^i</td>
<td>$</td>
<td>1094</td>
<td>1084</td>
<td>625</td>
<td>1046</td>
<td>480</td>
<td>606</td>
<td>709</td>
</tr>
<tr>
<td>precision(%)</td>
<td>99.63</td>
<td>91.94</td>
<td>99.36</td>
<td>99.52</td>
<td>100</td>
<td>72.75</td>
<td>49.88</td>
<td>47.64</td>
<td>38.38</td>
</tr>
<tr>
<td>recall(%)</td>
<td>95.71</td>
<td>94.76</td>
<td>59.24</td>
<td>99.05</td>
<td>45.30</td>
<td>57.44</td>
<td>62.03</td>
<td>99.43</td>
<td>98.95</td>
</tr>
</tbody>
</table>

Table 3.6.6: Clustering result of CURE for Pendigits data as $\alpha=0.4$ and $r=50$. For the last four columns, the corresponding relationship is four-to-one: four real clusters correspond to one detected cluster.

<table>
<thead>
<tr>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
<th>i=8</th>
<th>i=9</th>
<th>i=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>j=1</td>
<td>j=2</td>
<td>j=3</td>
<td>j=4</td>
<td>j=5</td>
<td>j=6</td>
<td>j=7</td>
<td>j=7</td>
<td>j=7</td>
<td>j=7</td>
</tr>
<tr>
<td>$</td>
<td>C_o^i</td>
<td>$</td>
<td>1143</td>
<td>1144</td>
<td>1055</td>
<td>1056</td>
<td>1055</td>
<td>1055</td>
<td>1143</td>
</tr>
<tr>
<td>$</td>
<td>C_s^j</td>
<td>$</td>
<td>897</td>
<td>758</td>
<td>462</td>
<td>835</td>
<td>125</td>
<td>28</td>
<td>4121</td>
</tr>
<tr>
<td>$</td>
<td>C_s^j \cap C_o^i</td>
<td>$</td>
<td>897</td>
<td>715</td>
<td>461</td>
<td>835</td>
<td>125</td>
<td>28</td>
<td>816</td>
</tr>
<tr>
<td>precision(%)</td>
<td>100</td>
<td>94.32</td>
<td>99.78</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>19.80</td>
<td>21.47</td>
<td>23.14</td>
</tr>
<tr>
<td>recall(%)</td>
<td>78.47</td>
<td>62.50</td>
<td>43.69</td>
<td>79.07</td>
<td>11.84</td>
<td>2.65</td>
<td>73.93</td>
<td>83.88</td>
<td>83.39</td>
</tr>
</tbody>
</table>

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. 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 $\alpha$ 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 \((\alpha, r)\) 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 3.6.4 shows the clustering results of CURE algorithm when \(\alpha\) 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 5.4.5 and 3.6.4. Our algorithm finds 4 of the 5 large clusters (with the sizes of 143, 52, 35 and 20) with the precisions of 0.9630, 0.6324, 0.6531 and 0.9091 and the recalls of 0.9091, 0.8269, 0.9143 and 0.5000, while CURE finds 3 of the 5 large clusters (with the sizes of 143, 77 and 52) with the precisions of 0.9583, 0.6119 and 0.9375 and the recalls of 0.8041, 0.5324 and 0.5769.

Figure 3.6.5(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, \(C_0^{i}\) for \(i = 1, 2, \ldots, 10\). Our algorithm detected eight clusters \(C_s^i\) for \(i = 1, 2, \ldots, 8\). The first six detected clusters, \(C_s^1\) through \(C_s^6\), correspond to \(C_0^1\) through \(C_0^6\) respectively. The seventh detected cluster, \(C_s^7\), corresponds to \(C_0^7\) and \(C_0^8\); and the last
detected cluster, $C_8^s$, corresponds to $C_9^o$ and $C_{10}^o$. Table 5.4.6 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. We set the cluster number parameter $k$ to 10, set the shrinking factor $\alpha$ 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 $(\alpha, r)$ 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 3.6.6 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 5.4.6 and 3.6.6. Our algorithm finds 8 clusters, while CURE finds 7 clusters. The average precision and average recall of our algorithm are 0.7231 and 0.7920, while the average precision and average recall of CURE are 0.6774 and 0.5851.

Figure 3.6.5(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.
3.6.4 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.

![Testing result of OPTICS for Wine data](image1)

Figure 3.6.6: Testing result of OPTICS for Wine data (a) without shrinking preprocessing (b) after shrinking preprocessing

![Testing result of OPTICS for Pendigits data](image2)

Figure 3.6.7: Testing result of OPTICS for Pendigits data (a) without shrinking preprocessing (b) after shrinking preprocessing

From Figure 3.6.6 we can see that after shrinking preprocessing, the cluster-ordering is much more significant than that without shrinking preprocessing. And the curve shown on Figure 3.6.6 (b) matches the ground truth (3 clusters of 59, 71, 48 data points) much better.
than the original curve on Figure 3.6.6 (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 3.6.7 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 3.6.7 shows the clustering results of CURE algorithm on Wine data after shrinking preprocessing when $\alpha$ is equal to 0.3, and r is 30. Comparing Table 3.6.7 to the original clustering result Table 3.6.2, 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>
<thead>
<tr>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>C_i^{\alpha}</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>C_i^{\alpha}</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>C_i^{\alpha} \cap C_i^{\alpha'}</td>
<td>$</td>
</tr>
<tr>
<td>precision(%)</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>recall(%)</td>
<td>81.25</td>
<td>60.56</td>
</tr>
</tbody>
</table>

Table 3.6.7: Clustering result of CURE for Wine data after shrinking preprocessing as $\alpha=0.3$ and r=30

**BIRCH:** We also used the implementation of BIRCH provided to us by the authors of [93]
to show how shrinking preprocessing will affect the performance of BIRCH on different data. Here we 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.

3.7 Conclusion and discussion

In this chapter, we proposed a new data preprocessing technique called shrinking which optimizes the inherent characteristic of distribution of data inspired by the Newton’s Universal Law of Gravitation. We applied the technique 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.
Chapter 4

A Shrinking-Based Dimension Reduction Approach

In this chapter, we present continuous research on data analysis based on our previous work on the shrinking approach. Shrinking[90] is a novel data preprocessing technique which optimizes the inner structure of data inspired by the Newton's Universal Law of Gravitation[70] in the real world. It can be applied in many data mining fields. Following our previous work on the shrinking method for multi-dimensional data analysis in full data space, we propose a shrinking-based dimension reduction approach which tends to solve the dimension reduction problem from a new perspective. In this approach data are moved along the direction of the density gradient, thus making the inner structure of data more
prominent. It is conducted on a sequence of grids with different cell sizes. Dimension reduction process is performed based on the difference of the data distribution projected on each dimension before and after the data-shrinking process. Those dimensions with dramatic variation of data distribution through the data-shrinking process are selected as good dimension candidates for further data analysis. This approach can assist to improve the performance of existing data analysis approaches. We demonstrate how this shrinking-based dimension reduction approach affects the clustering results of well known algorithms.

4.1 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. Shrinking is a novel data preprocessing technique which optimizes the inner structure of data inspired by the Newton’s Universal Law of Gravitation[70] in the real world. It can be applied in many data mining fields. In this chapter, we first give a brief introduction about our previous work on the shrinking concept formation and its application for clustering approaches in full-space; then, we propose a shrinking-based approach for the dimension reduction problem in multi-dimensional data analysis.
It is well acknowledged that in the real world a large proportion of data has irrelevant features which may cause a reduction in the accuracy of some algorithms. High dimensional data sets continue to pose a challenge to clustering algorithms at a very fundamental level. One of the well known techniques for improving the data analysis performance is the method of dimension reduction[6, 4, 73] in which data is transformed to a lower dimensional space while preserving the major information it carries, so that further processing can be simplified without compromising the quality of the final results. Dimension reduction is often used in clustering, classification, and many other machine learning and data mining applications.

There are several different ways in which the dimensions of a problem can be reduced. One approach is to identify important attributes based on input from domain experts. In other words, it is the optimal selection of a subset of attributes of existing dimensions (attributes). Another kind of commonly used approaches are projection methods in which the new projected dimensions are linear or un-linear combination of old dimensions. A popular approach is to use principal component analysis (PCA) through singular value decomposition (SVD)[49] for numerical attributes which defines new attributes (principal components or PCs) as mutually-orthogonal linear combinations of the original attributes. In information retrieval, latent semantic indexing uses SVD to project textual documents represented as document vectors. SVD is shown to be the optimal solution for a probabilistic model for
document/word occurrence. This kind of approaches has a major drawback in that the generated low dimensional subspace has no intuitive meaning to users. Random projections to subspaces have also been used.

4.1.1 Data shrinking

In this subsection, we will briefly introduce our previous work on the shrinking approach. Shrinking is a data preprocessing approach which optimizes the inner structure of data by moving each data point along the direction in which way it is most strongly attracted by other data points.

In the previous work, we proposed a shrinking-based approach for multi-dimensional data analysis which consists of three steps: data shrinking, cluster detection, and cluster evaluation and selection. This approach is based on grid separation of the data space. Since grid-based clustering approaches depend on the proper selection of grid-cell size, we used a technique called density span generation to select a sequence of grids of different cell sizes and perform the data-shrinking and cluster-detection steps based on these suitable grids.

In the data-shrinking step, each data point moves along the direction of the density gradient and the data set shrinks toward the inside of the clusters. Data points are “attracted” by their neighbors and move to create denser clusters. The neighboring relationship of the points in
the data set is grid-based. The data space is first subdivided into grid cells. Data points in sparse cells are considered to be noise or outliers and will be ignored in the data-shrinking process. Data-shrinking proceeds iteratively; in each iteration, we 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. The iterations terminate if the average movement of all points is less than a threshold or if the number of iterations exceeds a threshold.

Following the data-shrinking step, the cluster-detection step is performed in which neighboring dense cells are connected and a neighboring graph of the dense cells is constructed. A breadth-first search algorithm is applied to find connected components of the neighboring graph each of which is a cluster.

The clusters detected at multiple grid scales are compared by a cluster-wise evaluation measurement. We use the term compactness to measure the quality of a cluster on the basis of intra-cluster and inter-cluster relationships. Internal connecting distance (ICD) and external connecting distance (ECD) are defined 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. This definition of compactness is used to evaluate clusters detected at different scales and to then select the best clusters as the final result.
4.1.2 Proposed approach

In this chapter, we propose a shrinking-based dimension reduction approach for multi-dimensional data analysis to address the inadequacies of current clustering algorithms in handling multi-dimensional data. It tends to solve the dimension reduction problem from a new perspective.

In the proposed algorithm, data points are moved along the direction of the density gradient, leading to clusters which are condensed and widely-separated. This process is grid-based. Instead of choosing a grid with a fixed cell size, we use a sequence of grids of different cell sizes to perform our algorithm.

Dimension reduction process is performed based on the difference of the data distribution projected on each dimension through data-shrinking process. For those dimensions which make large contribution to the good results of data analysis (practically clustering process here), the alterations of the histogram variance of them through the data-shrinking process are significant. By evaluating the ratio of the histogram variances through data-shrinking process, good dimension candidates for further data analysis steps (e.g., clustering algorithms) can be picked out efficiently, and unqualified ones are discarded. It can improve the clustering results of existing clustering algorithms. We will demonstrate how the dimension reduction process will improve the performance of existing clustering algorithms.
in the experimental part.

The remainder of this paper is organized as follows. Section 4.2 discusses the details of dimension reduction process. Section 5.4 presents experimental results, and concluding remarks are offered in Section 7.1.

4.2 Dimension Reduction

In this section a shrinking-based approach to dimension reduction is proposed. In this approach, we consider optimal selection of a subset of existing dimensions for the purpose of easy interpretation.

4.2.1 Core concept

The main idea is that for those dimensions which make large contribution to the good results of data analysis (practically clustering process here), the alterations of the histogram variance (specified in the following description) of them through the data-shrinking process are significant. We select good dimension candidates for clustering based on the observation of the difference of the histogram status of each dimension.
Figure 4.2.1: Histogram variance of Ecoli data on each dimension (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction

The alteration of the histogram variances through the data-shrinking process on each dimension reflects the characteristic aspects of the data distribution on the dimension better than the histogram variance itself. Figure 4.2.1 shows an example for this case. From figure 4.2.1, we can see the dimension 3 and dimension 4 have much larger histogram variances than other dimensions both without and after shrinking-based dimension reduction process. However, these two dimensions give poor support for the following clustering process. The reason is that, on those dimensions, most part of the data are in a very narrow dense area, and just a few data points are far away from the dense area, which makes the histogram variance extremely large, although those dimensions are almost of no help to the capture of the characteristic of the data distribution. By evaluating the ratio of the histogram variance alteration on a dimension instead of the histogram variance itself, those dimensions are picked out and discarded efficiently.
4.2.2 Dimension reduction process

The shrinking-based dimension reduction approach is formalized as follows:

As mentioned in previous sections, we assume that the input $d$-dimensional dataset $\mathbf{X}$ is

$$\mathbf{X} = \{\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2, \ldots, \tilde{\mathbf{X}}_n\},$$

which is normalized to be within the hypercube $[0, 1]^d \subset \mathbb{R}^d$. Each data point $\tilde{\mathbf{X}}_i$ is a $d$-dimensional vector:

$$(4.2.1)\quad \tilde{\mathbf{X}}_i = \{X_{i1}, X_{i2}, \ldots, X_{id}\}.$$

Data-shrinking process is performed under various of grid scale candidate. For each grid scale candidate, for each dimension of $\mathbf{X}$, a histogram is set up bases on the current grid scale information:

$$(4.2.2)\quad H = \{H_1, H_2, \ldots, H_d\}.$$  

The number of segments on each dimension is not necessary the same. Let $\eta_i$ be the number of bins in the histogram on the $i$th dimension. We denote each histogram as:

$$(4.2.3)\quad H_i = \{H_{i1}, H_{i2}, \ldots, H_{i\eta_i}\},$$

in which $H_{ij}$ is a bin for the histogram. We denote the region of bin $H_{ij}$ as $[Min_{ij}, Max_{ij}]$.
for \( j = \eta_i \), or \([\text{Min}_{ij}, \text{Max}_{ij}]\) otherwise. The size of bin \( H_{ij} \) is the amount of data points whose \( i \)th attributes are in the region of \( H_{ij} \):

\[
(4.2.4) \quad |H_{ij}| = |\{X_l|\text{Min}_{ij} \leq X_{lj} \leq \text{Max}_{ij}\}|
\]

for \( j = \eta_i \), or

\[
(4.2.5) \quad |H_{ij}| = |\{X_l|\text{Min}_{ij} \leq X_{lj} < \text{Max}_{ij}\}|
\]

for \( j \neq \eta_i \).

Let \( \mu_{H_i} \) be the mean of the bin sizes of \( H_i \), and \( \sigma^2_{H_i} \) be the variance of the bin sizes of \( H_i \):

\[
(4.2.6) \quad \sigma^2_{H_i} = \frac{\sum_j (|H_{ij}| - \mu_{H_i})^2}{\eta_i}
\]

Let \( \tilde{\sigma}^2_{H_i} \) be the variance after data-shrinking process, we evaluate the variance difference between the original histogram status and after-shrinking histogram status by the ratio of the variances:

\[
(4.2.7) \quad \Delta \sigma^2_{H_i} = \frac{\tilde{\sigma}^2_{H_i}}{\sigma^2_{H_i}},
\]

The sum of the histogram variance status on all the dimensions under current grid scale
condition are calculated:

\[(4.2.8) \quad \Gamma = \sum_{i=1}^{d} \Delta \sigma_{H_i}^2\]

Suppose there are \(P\) reasonable grid scale candidates previously generated. We got \(P\) different histogram variance sums:

\[(4.2.9) \quad S = \{S_1, S_2, \ldots, S_P\}.\]

Under some grid scales, data can not be properly shrunk via data shrinking process. In another word, those grid scales can not help make the data distribution more piercing. In such cases, the variance difference are not prominent. Those grid scales are discarded.

For those grid scales selected according to the variance difference sums, dimensions of significant variance change through data shrinking process are selected as good candidates for clustering process, based on the integrated variance differences under these grid scales.

Under each selected grid scale \(\lambda\), we sort dimensions \(D_1, D_2, \ldots, D_d\) in descending order according to \(\Delta \sigma_{H_i}^2\). Suppose the dimension list after sorting is \(\hat{D}_1, \hat{D}_2, \ldots, \hat{D}_d\), we select the first several dimensions. The cut on the dimension list is performed as follows. To keep most valuable dimensions, the second half of the ordered dimension list is checked, and the cut spot is set on the first sharp descent dimension. We will demonstrate the dimension cut process in the experimental part.

For each grid scale candidate, an ordered dimension list based on the variance difference is
generated. The ultimate selection of valuable dimensions is based on the integration of the dimension selections on these qualified grid scales.

4.2.3 Time and space analysis

Throughout the dimension reduction process, we need to keep track of the histogram information of each dimension for each grid scale candidate. Suppose the number of the grid scale candidates is $P$, the maximum bin amount of a dimension is $\eta_{\text{max}}$, and the dimensionality is $d$. The dimension reduction process occupies $O(Pd\eta_{\text{max}})$ space.

Under a certain grid scale, the time of the dimension sorting based on the variance differences is $O(d\log d)$. So the altogether sorting time of dimensions for all suitable grid scales is $O(Pd\log d)$.

4.3 Experiments

We conducted comprehensive experiments 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. Experiments are conducted to demonstrate how the shrinking-based dimension reduction approach can benefit well known algorithms. 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 that it will help improve the performance of existing clustering algorithms such as OPTICS and BIRCH as well as data visualization tools such as VizCluster[57].

4.3.1 Data sets

The three data sets were obtained from UCI Machine Learning Repository [10].

Data Set 1 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 with the sizes of 59, 71 and 48.

Data Set 2 Ecoli data contains data regarding Protein Localization Sites. This data set is made up of 336 instances, with each instance having seven features. It contains 8 clusters with the sizes of 143, 77, 52, 35, 20, 5, 2 and 2.

Data Set 3 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 with the sizes of 1143, 1144, 1055, 1056, 1055, 1055, 1143, 1055, 1144 and 1142.

4.3.2 Shrinking-based dimension reduction approach on real data sets

Wine Data We first performed the data-shrinking process on the Wine data. Table 4.3.1 shows the histogram variance differences of Wine data before and after the data-shrinking process for a certain grid scale. We can see from Table 4.3.1 that the twelfth dimension has the most dramatic histogram variance change. Figure 4.3.1 shows the cut of the integrated ordered variance differences list on multiple scales for Wine data. Seven dimensions are selected for further processing.

<table>
<thead>
<tr>
<th>dimension</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{H_i}^2$</td>
<td>290.2</td>
<td>619.4</td>
<td>1231.4</td>
<td>847.0</td>
<td>867.4</td>
<td>254.2</td>
<td>514.6</td>
<td>373.8</td>
<td>654.6</td>
<td>620.2</td>
<td>494.6</td>
<td>155.8</td>
<td>449.4</td>
</tr>
<tr>
<td>$\tilde{\sigma}_{H_i}^2$</td>
<td>776.6</td>
<td>935.8</td>
<td>4009.4</td>
<td>1879.8</td>
<td>2877.4</td>
<td>834.6</td>
<td>851.0</td>
<td>735.4</td>
<td>1718.6</td>
<td>861.8</td>
<td>1719.8</td>
<td>724.6</td>
<td>735.8</td>
</tr>
<tr>
<td>$\gamma_{\sigma_{H_i}}$</td>
<td>2.67</td>
<td>1.51</td>
<td>3.25</td>
<td>2.21</td>
<td>3.31</td>
<td>3.28</td>
<td>1.65</td>
<td>2.02</td>
<td>2.62</td>
<td>1.38</td>
<td>3.47</td>
<td>4.65</td>
<td>1.63</td>
</tr>
</tbody>
</table>

Table 4.3.1: histogram variance differences of Wine data for a certain grid scale

<table>
<thead>
<tr>
<th>dimension</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_{H_i}^2$</td>
<td>421.4</td>
<td>898.4</td>
<td>9509.0</td>
<td>10094.0</td>
<td>1264.8</td>
<td>415.2</td>
<td>697.2</td>
</tr>
<tr>
<td>$\tilde{\sigma}_{H_i}^2$</td>
<td>2276.8</td>
<td>2389.0</td>
<td>9509.0</td>
<td>10094.0</td>
<td>2453.2</td>
<td>1646.6</td>
<td>1706.4</td>
</tr>
<tr>
<td>$\gamma_{\sigma_{H_i}}$</td>
<td>5.40</td>
<td>2.65</td>
<td>1.0</td>
<td>1.0</td>
<td>1.93</td>
<td>3.96</td>
<td>2.44</td>
</tr>
</tbody>
</table>

Table 4.3.2: histogram variance differences of Ecoli data for a certain grid scale
CHAPTER 4. A SHRINKING-BASED DIMENSION REDUCTION APPROACH

<table>
<thead>
<tr>
<th>dimension</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma^2_{H_i} \times 10^6$</td>
<td>1.05</td>
<td>8.34</td>
<td>6.22</td>
<td>6.66</td>
<td>2.93</td>
<td>1.39</td>
<td>8.31</td>
<td>1.82</td>
<td>6.34</td>
<td>1.27</td>
<td>2.05</td>
<td>1.21</td>
<td>1.67</td>
<td>2.05</td>
<td>2.36</td>
<td>4.62</td>
</tr>
<tr>
<td>$\bar{\sigma}^2_{H_i} \times 10^6$</td>
<td>1.97</td>
<td>10.29</td>
<td>72.60</td>
<td>7.58</td>
<td>23.67</td>
<td>2.10</td>
<td>440.07</td>
<td>13.71</td>
<td>35.55</td>
<td>2.98</td>
<td>6.25</td>
<td>2.11</td>
<td>6.86</td>
<td>4.07</td>
<td>3.12</td>
<td>6.17</td>
</tr>
<tr>
<td>$\gamma \sigma^2_{H_i}$</td>
<td>1.87</td>
<td>1.23</td>
<td>11.6</td>
<td>1.13</td>
<td>8.06</td>
<td>1.50</td>
<td>52.9</td>
<td>7.49</td>
<td>5.60</td>
<td>2.35</td>
<td>3.03</td>
<td>1.73</td>
<td>4.08</td>
<td>1.99</td>
<td>1.32</td>
<td>1.33</td>
</tr>
</tbody>
</table>

Table 4.3.3: histogram variance differences of Pendigits data for a certain grid scale

![Figure 4.3.1: The cut of the variance differences list on Wine data](image)

**Ecoli Data** We then performed the data-shrinking process on the Ecoli data. Table 4.3.2 shows the histogram variance differences of Ecoli data before and after the data-shrinking process for a certain grid scale. We can see from Table 4.3.2 that the first dimension has the most dramatic histogram variance change. Figure 4.3.2 shows the cut of the integrated ordered variance differences list on multiple scales for Ecoli data. Five dimensions are selected for further processing.

**Pendigits Data** We also performed the data-shrinking process on the Pendigits data. Table 4.3.3 shows the histogram variance differences of Pendigits data before and after the data-shrinking process for a certain grid scale. We can see from Table 4.3.3 that the seventh dimension has the most dramatic histogram variance change. Figure 4.3.3 shows the cut of
the integrated ordered variance differences list on multiple scales for Pendigits data. Eleven dimensions are selected for further processing.

4.3.3 Testing results of existing clustering algorithms

In this section we will demonstrate how the shrinking-based dimension reduction approach improves the performance of well known algorithms such as OPTICS and BIRCH, as well
as data visualization tools such as VizCluster.

**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. We set the parameter values for OPTICS just to be “large” enough to yield a good result.

Figure 4.3.4: Testing result of OPTICS for Wine data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction

Figure 4.3.5: Testing result of OPTICS for Ecoli data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction

Figure 4.3.4 shows the testing results of OPTICS on Wine data before and after the shrinking-based dimension reduction process. We can see that after shrinking-based dimension reduction process, the clusters shown in the curve of Figure(b) are much more clear than the
Figure 4.3.6: Testing result of OPTICS for Pendigits data (a) without shrinking-based dimension reduction (b) after shrinking-based dimension reduction

Figure 4.3.7: Testing result of VizCluster for Wine data without shrinking-based dimension reduction

original one(a).

Figure 4.3.5 shows the testing results of OPTICS on Ecoli data before and after the shrinking-based dimension reduction process. The clusters shown in the curve of Figure(b) are also more clear than the original one(a).

Figure 4.3.6 shows the testing results of OPTICS on Pendigits data before and after the shrinking-based dimension reduction process. Again the performance with shrinking-based dimension reduced data are better than the original one.
**VizCluster:** VizCluster[57] is an interactive visualization tool for multi-dimensional data. It combines the merits of both multi-dimensional scatter-plot and parallel coordinates. Integrated with useful features, it can give a simple, fast, intuitive and yet powerful view of the data set. Due to the space limitation, here we just present the testing results on Wine data and Ecoli data. Figures 4.3.7, 4.3.8, 4.3.9 and 4.3.10 show the testing results on these two data sets respectively. Different shapes of the points present different cluster id
From Figures 4.3.7 and 4.3.8 we can see that the visualization result of VizCluster on the shrinking-based dimension reduced Wine data is much better than that on the original one.

From Figures 4.3.9 and 4.3.10 we can see that the visualization result of VizCluster on the shrinking-based dimension reduced Ecoli data is also better than that on the original one.

**BIRCH:** We also tested how the shrinking-based dimension reduction approach affects the performance of BIRCH on different data. Due to the space limitation, here we just show the testing result on ecoli data mentioned in the 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. Our test includes two steps. In the first step, we applied the BIRCH algorithm directly on the data, resulting in 8 clusters with the sizes of 133, 93, 74, 24, 6, 3, 2 and 1. In the second step, we applied BIRCH again on the data after the shrinking-based dimension reduction process,
and get 8 clusters with the sizes of 143, 101, 76, 7, 4, 2, 1 and 1. From the comparison of the two different clustering results, we can see that the major clusters generated after the shrinking-based dimension reduction process match the ground truth better than those generated from the original BIRCH algorithm.

4.4 Conclusion and discussion

In this chapter, we presented a shrinking-based dimension reduction approach for multi-dimensional data. We select good dimension candidates for further data analysis based on the observation of the alteration of the histogram variance of each dimension through the data-shrinking process. We demonstrated the effectiveness and efficiency of our approach by the demonstration of tests on real data sets.

Data analysis methods still pose many open issues. Shrinking-based approaches rely on the selection of grid scales to a certain degree. Improvement of the grid scale acquisition approach will greatly benefit the whole shrinking concept and implementation. This is one of our primary further researches.
Chapter 5

A Cluster-Outlier Iterative Detection Approach

Nowadays many data mining algorithms focus on clustering methods. There are also a lot of approaches designed for outlier detection. We observe that, in many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise. Thus, it is necessary to treat clusters and outliers as concepts of the same importance in data analysis. In this chapter, we present COID, a cluster-outlier iterative detection algorithm, tending to detect the clusters and outliers in another perspective for noisy data sets. In this algorithm, clusters are detected and adjusted according to the intra-relationship within clusters and the inter-relationship between clusters and outliers,
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached. This data processing algorithm can be applied in many fields such as pattern recognition, data clustering and signal processing. Experimental results demonstrate the advantages of our approach.

5.1 Introduction

More and more large quantities of multi-dimensional data need to be clustered and analyzed. Many new clustering, outlier detection and cluster evaluation approaches are presented in the last a few years. Nowadays a lot of real data sets are noisy, which makes it more difficult to design algorithms to process them efficiently and effectively.

Cluster analysis is used to identify homogeneous and well-separated groups of objects in data sets. It plays an important role in many fields of business and science.

An outlier is a data point that does not follow the main characteristics of the input data. Outlier detection is concerned with discovering the exceptional behaviors of certain objects. It is an important branch in the field of data mining with numerous applications, including credit card fraud detection, discovery of criminal activities, discovery of computer intrusion, and etc. and in some sense it is at least as significant as cluster detection. There are numerous studies on outlier detection. D. Yu etc. [91] proposed an outlier detection
approach termed FindOut as a by-product of WaveCluster [76] which removes the clusters from the original data and thus identifies the outliers. E. M. Knorr etc. [54] detected a distance-based outlier which is a data point with a certain percentage of the objects in the data set having a distance of more than $d_{\text{min}}$ away from it. S. Ramaswamy etc. [68] further extended it based on the distance of a data point from its $k^{th}$ nearest neighbor and identified the top $n$ points with largest $k^{th}$ nearest neighbor distances as outliers. M. M. Breunig etc. [19] introduced the concept of local outlier and defined local outlier factor (LOF) of a data point as a degree of how isolated the data point is with respect to the surrounding neighborhood. Aggarwal etc. [5] considered the problem of outlier detection in subspace to overcome dimensionality curse.

Our approach is different from the previous clustering and outlier detection methods in that we tried to detect and adjust the set of clusters and outliers according to the intra-relationship in the set of clusters and the set of outliers, as well as the inter-relationship between clusters and outliers.

There are several criteria for quantifying the similarity (dissimilarity) of the clusters. [14, 34, 50]. ROCK[34] measures the similarity of two clusters by comparing the aggregate inter-connectivity of two clusters against a user-specified static inter-connectivity model. Chameleon [50] measures the similarity of two clusters based on a dynamic model. Two clusters are merged only if the inter-connectivity and closeness (proximity) between two
clusters are highly relative to the internal inter-connectivity of the clusters and closeness of items within the clusters.

Many approaches [21, 59] have been proposed for evaluating the results of a clustering algorithm. M. Halkidi et al. [59] 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. The index incorporates criteria addressing compactness, separation and density. C.F. Chen et al. [21] introduced a fuzzy validity function to measure the overall average compactness and separation of the fuzzy partition. The average compactness is estimated by the deviation of data points from the center of each cluster, and the separation of the partition is represented by the distance between cluster centers. These clustering validity measurements evaluate clustering algorithms by measuring the overall quality of the clusters.

We observe that, in many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise. Thus, it is necessary to treat clusters and outliers as concepts of the same importance in data analysis.

Another fundamental problem in data analysis field is that clusters and outliers are detected mostly based on the information of the features of data sets, and the results are compared to ground truth of natural clusters and outliers. However, in many cases in the real world, the ground truth and the information of features of the real data sets do not match each other.
very well, and good results are hard to achieve even using dimension reduction approaches. It is another motive for the development of our algorithm. We tend to detect the clusters and outliers in another perspective, not only relying on the features of the data sets, but also exploiting the relationship between clusters and outliers in a computable way.

In this paper, we present a cluster-outlier iterative detection (simplified as COID in the following content) algorithm for noisy multi-dimensional data set. In this algorithm, clusters are detected and adjusted according to the intra-relationship among clusters and the inter-relationship between clusters and outliers, and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached.

The remainder of this paper is organized as follows. Section 5.2 presents the formalization and definitions of the problem. Section 5.3 describes the cluster-outlier iterative detection (COID) algorithm. Section 5.4 presents experimental results, and concluding remarks are offered in Section 7.1.

5.2 Problem Definition

The concepts of cluster and outlier are related to each other. Real world data don’t necessarily have natural clusters at all. And for those which do have clusters, there are seldom
the cases in reality that the data objects (data points) in the data all belong to some natural cluster. In other words, there are normally outliers existing in the data. One of the aspects of the qualities of clusters and outliers is reflected by how much diversity they have inside and have to each other. Clusters and outliers are concepts whose meanings are inseparable to each other. Thus, it is necessary to treat clusters and outliers as the same important concepts in the data processing. Equal treatment to clusters and outliers can benefit applications in many fields.

The cluster-outlier iterative detection (COID) problem is formalized as follows. In order to describe our approach we shall introduce a few notation and definitions. Let \( n \) denote the total number of data points and \( d \) be the dimensionality of the data space. Let the input \( d \)-dimensional dataset be \( X \)

\[
X = \{\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_n\},
\]

which is normalized to be within the hypercube \([0, 1]^d \subset \mathbb{R}^d\). Each data point \( \tilde{X}_i \) is a \( d \)-dimensional vector:

\[
(5.2.1) \quad \tilde{X}_i = [x_{i1}, x_{i2}, \ldots, x_{id}].
\]

Our main goal is to refine and improve the clustering and outlier detection results of clustering algorithms. In this paper, for the sake of consistency and convenient interpretation of our approach, we develop our own clustering algorithm and perform the refinement and improvement process on its results. However, we demonstrate how COID can help improve
the testing results of other algorithms as well in the experimental part. Based on the input of the initial (or “rough”) clustering and outlier detection results, the COID process adjusts and modifies the cluster-outlier division of the data set iteratively until a certain termination condition is reached.

According to the input of the initial cluster-outlier division of a data set, we perform the COID algorithm in an iterative way. In a given iteration step, we assume the current number of clusters is $k_c$, and the current number of outliers is $k_o$. The set of clusters is $\mathcal{C} = \{C_1, C_2, ..., C_t\}$, and the set of outliers is $O = \{O_1, O_2, ..., O_t\}$. Here we use the term compactness to measure the quality of a cluster.

### 5.2.1 Distance metric selection

It is crucial to define a proper distance metric for a specific data mining problem. We use $d(p_1, p_2)$ to represent the distance between data points $p_1$ and $p_2$ under a certain distance metric. In a high dimensional space the data are usually sparse, and widely used distance metric such as Euclidean distance may not work well as dimensionality goes higher.

Recent research [15] shows that in high dimensions nearest neighbor queries become unstable: the difference of the distances of farthest and nearest points to some query point does not increase as fast as the minimum of the two. Aggarwal, etc. [3, 41] viewed the
dimensionality curse from the point of view of the distance metrics and argued that for the
commonly used $L_K$ norm, the problem of meaningfulness in high dimensionality is sensi-
tive to the value of K: it worsens faster with increasing dimensionality for high values of
K. In this case, the Manhattan distance metric ($L_1$ norm) is consistently more preferable
than the Euclidean distance metric ($L_2$ norm) for high dimensional data mining applica-
tions. They also exploited the research in fractional distance metrics [3] which can further
improve the effectiveness of standard clustering algorithms. Based on the current research,
we prefer $L_{0,1}$ to $L_2$ metric in our implementations.

5.2.2 The compactness of a cluster

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 [21, 59], 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.

Given a set of clusters $\mathcal{C} = \{C_1, C_2, ..., C_n\}$, an easy way to define compactness(CPT) can be
the closeness measurement of the data points in $C_i$ to the centroid of $C_i$:

$$CPT(C_i) = \frac{\sum_{p \in C_i} d(p, m_{c_i})}{|C_i|},$$

(5.2.2)

where $m_{c_i}$ is the centroid of Cluster $C_i$, $p$ is any data point in Cluster $C_i$, $|C_i|$ is the number
of data points in $C_i$, and $d(p,m_{c_i})$ is the distance between $p$ and $m_{c_i}$. The centroid $m_{c_i}$ of the cluster is the algebraic average of all the points in the cluster: $m_{c_i} = \sum_{p \in C_i} p / |C_i|$. However, such a definition does not consider the shape and data distribution of the cluster, as well as the relationship between the cluster and other clusters.

Here we define the compactness as follows:

**Definition 1:** For a connected subset $S$ of $V$, let $\text{MST}(S)$ be a minimum spanning tree of the minimal subgraph containing $S$. The **internal connecting distance** (ICD) of $S$, denoted as $\text{ICD}(S; G, w)$, is defined as the length of a longest edge of $\text{MST}(S)$. The **external connecting distance** (ECD) of $S$, denoted as $\text{ECD}(S; G, w)$, is defined as the length of a shortest edge connecting $S$ and $V - S$. The **compactness** of $S$, denoted as $\text{Compactness}(S; G, w)$, is defined as

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

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

In the following we use $\text{CPT}(S)$ to denote $\text{Compactness}(S; G, w)$. 
5.2.3 The diversities of data groups

We use the term *diversity* to describe the difference between two clusters, the difference between two outliers, and the one between a cluster and an outlier.

The way to denote the diversity between an outlier and a cluster is similar to the problem of the distance measurement between a query point and a cluster which is well studied in the data mining field. An advanced approach is presented in [25] in which the authors defined an optimized distance between a query point \( q \) and a cluster \( C \) using a weighted combination of \( d_{\text{min}} \) (the minimum distance between \( q \) and the bounding hypersphere of \( C \)) and \( d_{\text{avr}} \) (the average distance between \( q \) and the bounding hypersphere of \( C \)) since they claimed that neither \( d_{\text{min}} \) nor \( d_{\text{avr}} \) has uniformly good estimation even in a single structure because the clusters do not have the same data distribution.

\[
\begin{align*}
d_{\text{avr}}(q, \text{Sphere}(c, r)) &= d(q, c) \\
d_{\text{min}}(q, \text{Sphere}(c, r)) &= \begin{cases} 
  d(q, c) - r & \text{if } d(q, c) > r, \\
  0 & \text{otherwise},
\end{cases}
\end{align*}
\]

(5.2.4)

where \( r \) is the radius of the smallest sphere \( S \) centered at \( C \) covering data points in \( C \). They use the *density* \( \rho \) to set up the weights: \( \frac{\rho}{\rho + 1} \) for \( d_{\text{min}} \) and \( \frac{1}{\rho + 1} \) for \( d_{\text{avr}} \). The *density* \( \rho \) is computed as:

\[
\rho = \frac{\text{number of points in } C}{\text{volume of } S} = \frac{\text{number of points in } C}{r^\log d}.
\]
However, such density definition does not necessarily denote the inner structure of a cluster accurately. Figure 5.2.1 shows an example of two clusters in 2-dimensional data space. From Figure 5.2.1 we can see these two clusters have the same density according to the density definition mentioned above since they have the same number of data points and the same r. However, from the observation of the figure we can easily tell that the cluster on the right side is actually more compact than the cluster on the left side.
To solve this kind of problems, we can use our compactness (CPT) concept of the cluster instead of the density to set up the weights for the distance measurement. The diversity between a cluster $C$ and an outlier $O$ is defined as:

$D_1(C, O) = w_1 \cdot d_{\text{min}}(O, C) + w_2 \cdot d_{\text{avr}}(O, C)$

where $w_1 = \frac{1}{\text{CPT}(C) + 1}$, $w_2 = \frac{\text{CPT}(C)}{\text{CPT}(C) + 1}$, $d_{\text{avr}}(O, C) = d(O, m_c)$ and $d_{\text{min}}(O, C) = \max(d(O, m_c) - r_{\text{max}}, 0)$ where $r_{\text{max}}$ is the maximum distance of the data points in $C$ from its centroid.

However, this definition favors clusters of spherical shapes in some cases. Figure 5.2.2 show an example in which the two ellipse-like clusters have the compactness, same $r_{\text{max}}$, same $d(O, m_c)$ thus the same $d_{\text{avr}}(O, C)$ and $d_{\text{min}}(O, C)$ with the outlier $O$. Hence the distances from the outliers to the clusters based on the definition described above are the same. However, since the orientations of the two clusters are different, the distances from the outliers to the clusters should also be different.

Actually most real data contains different kinds of skews in the distribution of the data points in it, and the original coordinate system might not be suitable for the distribution of the data. Singular Value Decomposition (SVD)[49] is a well known technique which transforms the data into a new coordinate system in which the correlations in the data are minimized. It constructs a $d \times d$ covariance matrix for the data set, and derives the eigenvectors of the matrix which define an orthonormal system along which the second order correlations in the data are removed. We are only interested in the acquirement of the eigenvector
with the maximum spread which retain the most information which distinguishes the data points from one another. Once we get the eigenvector, we calculate the angle between the eigenvector and the vector connecting the cluster centroid and the outlier \( O \). We map the angle into the range of \([0,90]\). The larger the angle is, the farther the outlier is away from the cluster assumed other parameters are all the same. We use the dot product to calculate \( |\cos\theta| \). Given two vectors \( \vec{V}_1 \) and \( \vec{V}_2 \), the dot product of these two vectors can be defined as: 
\[
\vec{V}_1 \cdot \vec{V}_2 = |\vec{V}_1||\vec{V}_2|\cos\theta.
\]
So we can compute \( |\cos\theta| = \frac{|\vec{V}_1 \cdot \vec{V}_2|}{|\vec{V}_1||\vec{V}_2|} \). We modify the definition as follows.

**Definition 2:** The diversity between a cluster \( C \) and an outlier \( O \) is defined as:

\[
D_1(C, O) = \frac{w_1 \cdot d_{\min}(O, C) + w_2 \cdot d_{\text{avr}}(O, C)}{1 + |\cos\theta|}
\]

where \( w_1 = \frac{1}{\text{CPT}(C)+1}, w_2 = \frac{\text{CPT}(C)}{\text{CPT}(C)+1}, d_{\text{avr}}(O, C) = d(O, m_c), d_{\text{min}}(O, C) = \max(d(O, m_c) - r_{\text{max}}, 0) \) where \( r_{\text{max}} \) is the maximum distance of the data points in \( C \) from its centroid, and \( \theta \) is the angle between the eigenvector of cluster \( C \) corresponding to the largest eigenvalue and the vector connecting \( m_c \) and \( O \). The criteria for setting the weights \( w_1 \) and \( w_2 \) are similar to those in [25].

According to this definition, the distance between the outlier and the cluster of case a in Figure 5.2.2 should have a larger value than the one in case b.
The distance between two clusters may not always represent their diversity accurately. Figure 5.2.3 shows an example of the relationship between clusters. Clusters $C_{11}$ and $C_{12}$ are sparser than clusters $C_{21}$ and $C_{22}$. From the global point of view, the former cluster pair ($C_{11}$ and $C_{12}$) is more likely to be merged into a larger cluster than the latter one. However, neither their minimum distance nor their average distance is different from each other ($d_{\text{min}}(C_{11}, C_{12}) = d_{\text{min}}(C_{21}, C_{22})$ and $d_{\text{avr}}(C_{11}, C_{12}) = d_{\text{avr}}(C_{21}, C_{22})$).

Similar cases are mentioned in the previous research. For example, the algorithm ROCK[34] proposes the concept of links two data points $p_i$ and $p_j$ to measure the similarity/proximity between a pair of data points, and it tends to maximize the sum of $\text{link}(p_q, p_r)$ for data
point pairs $p_q, p_r$ belonging to the same cluster, and minimize the sum of $\text{link}(p_q, p_s)$ for $p_q, p_s$ in different clusters at the same time. Then it defines goodness measure for merging two clusters $C_i$ and $C_j$:

$$g(C_i, C_j) = \frac{\text{link}[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)}} - n_j^{1+2f(\theta)}$$

where $C_i, C_j$ denote cluster $i$ and $j$ of sizes $n_i$ and $n_j$, respectively, $\theta$ is a user-defined parameter used to control how close a pair of points must be in order to be considered neighbors, $f(\theta)$ is some function of $\theta$, and $\text{link}[C_i, C_j]$ is the number of cross links between these two clusters.

Another algorithm Chameleon[50] determines the pair of most similar sub-clusters by taking into account both the inter-connectivity as well as the closeness of the clusters. The relative inter-connectivity between a pair of clusters $C_i$ and $C_j$ is defined as the absolute inter-connectivity $EC_{C_i,C_j}$ (weighted sum of the edges that connect vertices in $C_i$ to vertices in $C_j$) between $C_i$ and $C_j$ normalized with respect to the internal inter-connectivity $EC_{C_i}$ and $EC_{C_j}$ (the weighted sum of edges that partition a cluster into two roughly equal parts) of the two clusters $C_i$ and $C_j$:

$$RI(C_i, C_j) = \frac{|EC_{C_i,C_j}|}{|EC_{C_i}| + |EC_{C_j}|}.$$

And the relative closeness between a pair of clusters $C_i$ and $C_j$ is defined as the absolute closeness between $C_i$ and $C_j$ normalized with respect to the internal closeness of the two
clusters $C_i$ and $C_j$:

\[(5.2.9) \quad RC(C_i, C_j) = \frac{\overline{S}_{EC(C_i, C_j)}}{|C_i| + |C_j|} \frac{\overline{S}_{EC_i}}{|C_i|} + \frac{|C_j|}{|C_i| + |C_j|} \frac{\overline{S}_{EC_j}}{|C_j|} \]

where $\overline{S}_{EC_i}$ and $\overline{S}_{EC_j}$ are the average weights of the edges that belong in the min-cut bisector of clusters $C_i$ and $C_j$, respectively, and $\overline{S}_{EC(C_i, C_j)}$ is the average weight of the edges that connect vertices in $C_i$ to vertices in $C_j$.

However, the concepts presented in ROCK cater more to data sets with categorical attributes. As for Chameleon, since it is based on the weighted k-nearest neighbor graph of a data set, for each data point, its k-nearest neighbors need to be acquired, so the computation time cost is quite high, especially for high dimensional data sets. And the computation of a bisection is also non-trivial.

Here we apply a simplified criterion which integrates the compactness concept into the diversity measurement of two clusters which represents how compact the data points inside the cluster is.

**Definition 3:** The diversity between two clusters $C_1$ and $C_2$ is defined as:

\[(5.2.10) \quad D_2(C_1, C_2) = \frac{d(C_1, C_2)}{CPT(C_1) + CPT(C_2)} \times (1 + |\cos \theta|) \]

where $d(C_1, C_2)$ can be either the average distance between the two clusters or the minimum distance between them. Here we simply apply the former one $d(m_{C_1}, m_{C_2})$. $\theta$ is the
angle between the two eigenvectors corresponding to the two largest eigenvalues of \( C_1 \) and \( C_2 \), respectively. The larger the value of \( D_2(C_1, C_2) \) is, the larger diversity the clusters \( C_1 \) and \( C_2 \) have to each other.

In this case, for the situation shown in Figure 5.2.3, clusters \( C_{11} \) and \( C_{12} \) will be chosen for merger other than clusters \( C_{21} \) and \( C_{22} \) although they are sparser than the latter ones.

**Definition 4:** The diversity between two outliers \( O_1 \) and \( O_2 \) is defined as:

\[
D_3(O_1, O_2) = d(O_1, O_2)
\]

### 5.2.4 The qualities of data groups

In this subsection we define the quality of a cluster and the quality of an outlier.

We propose a novel way to define the quality of a cluster \( C \). The quality of \( C \) is reflected not only by the diversity between it and other clusters (how far away and different they are from each other), but also by the diversity between it and outliers. In other words, if \( C \) is near some outliers, its quality should certainly be impacted, because outliers are supposed to be far away from any cluster. So we take consideration of both the diversity between
Definition 5: We measure the quality of a cluster $C$ as:

\[
Q_c(C) = \frac{\sum_{C' \in C, C' \neq C} D_2(C, C')} {k_c - 1} + \frac{\sum_{O \in O} D_1(C, O)} {k_o}
\]

The larger $Q_c(C)$ is, the better quality cluster $C$ has.

Similarly, the quality of an outlier $O$ is reflected not only by the diversity between it and clusters, but also by the diversity between it and other outliers. The farther distances it has from other outliers and clusters, the better quality it should obtain.

Definition 6: We measure the quality of an outlier $O$ as:

\[
Q_o(O) = \frac{\sum_{O' \in O, O' \neq O} D_3(O, O')} {k_o - 1} + \frac{\sum_{C \in C} D_1(C, O)} {k_c}
\]

The larger $Q_o(O)$ is, the better quality outlier $O$ has.
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

5.3 Algorithm

The main goal of the COID algorithm is to mine the optimal set of clusters and outliers for the input data set. As we mentioned in the previous sections, in our approach, clusters and outliers of multi-dimensional data are detected, adjusted and improved iteratively. Clusters and outliers are closely related and they affect each other in a certain way. The basic idea of our algorithm is that clusters are detected and adjusted according to the intra-relationship within clusters and the inter-relationship between clusters and outliers, and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached. This analysis approach for multi-dimensional data can be applied in many fields such as pattern recognition, data clustering and signal
processing. The overall pseudocodes for the algorithm is given in Figure 5.3.1.

The algorithm proceeds in two phases: an initialization phase and an iterative phase. In the initialization phase, we find the centers of clusters and locations of outliers. The problem of finding cluster centers has been extensively investigated. For example, in [29, 18], the authors proposed a procedure for computing a refined starting condition from a given initial one that is based on an efficient technique for estimating the modes of a distribution. Here we applied some strategy similar to the greedy technique proposed in [33] which tries to iteratively pick up surrogate cluster centers (medoids). We also developed some criteria to determine the qualities (possibilities) of the selected medoids as centers of clusters. In the iterative phase, we refine the set of clusters and outliers gradually by optimally exchanging some outliers and some boundary data points of the clusters. Each phase is detailed in the following.

5.3.1 Initialization phase

In the initialization phase, we first find the initial set of medoids. In the next step we dispatch data points to their nearest medoids, forming data subsets associated with medoids. Then we exploit some approaches to determine whether a data subset is a cluster or a group of outliers. Following is each detailed step.
Algorithm COID \((k: \text{No. of Clusters})\)

Begin
1. Initialization Phase
   Repeat
   Const1 and Const2 are two proportion constants to \(k\), Const1 > Const2;
   RandomSize1 = Const1 \(\cdot k\);
   RandomSize2 = Const2 \(\cdot k\);
   \(RS_1\) = random sample with the size of RandomSize1;
   \(RS_2\) = FindKMedoids(\(RS_1\), RandomSize2);
   \{Assign data points to medoids to form medoid-associated sets\}
   \(E\) ← DispatchDataPoints(\(RS_2\)); \{\(E\) is set of initial data division;\}
   \{Determine the characteristics of the medoid-associated sets\}
   \(\{C\text{ and }\emptyset\}\) ← ClusterOrOutlier();
   Until(\(|C| \geq k|\)

2. Iterative Phase
   \{Merge the clusters according to the input cluster number \(k\)\}
   \(C\) ← MergeCluster(\(C\));
   Repeat
   \{Find the nearest cluster for each outlier\}
   For each outlier \(o\) ∈ \(\emptyset\) do
   Begin
   Find its nearest cluster ∈ \(C\)
   End
   Sort current set of clusters in ascending order based on their qualities;
   Sort current set of outliers in ascending order based on their qualities;
   \{Reorganize the structure of clusters and outliers\}
   ExchangeClusterAndOutlier();
   \(O'\) is the set of outliers with worst qualities;
   BDP is the set of boundary data points with worst qualities;
   \(U = O' \cup BDP\);
   Until(\(U\) is stable or iteration number ≥ 3)
End.

Figure 5.3.1: Algorithm: COID
Acquirement of medoids

It is critical to find medoids which can approximate the centers of different clusters for our approach. Since outliers normally exist in real data sets, it is not effective to apply the greedy technique of [33] directly to find $k$ medoids ([4]). Similar to the method proposed in [4], we try to find a *superset* of a good set of $k$ medoids. Also we develop a novel approach to determine the set of $k$ medoids from its superset.

Similar to [4], first we choose a random set $RS_1$ of data points from the original data set with the size of RandomSize1 which is proportional to the required cluster number $k$. Then we apply the greedy algorithm in [33] to find another random set $RS_2$ from $RS_1$ with the size of RandomSize2 which is also proportional to $k$, and RandomSize1 > RandomSize2. By applying the greedy algorithm on $RS_2$, the efficiency of the algorithm is greatly improved, and the number of outliers generated by the algorithm is largely reduced.

Dispatching data points

Once we get the smaller random set $RS_2$ of medoids, we shall find a way to determine which medoids are in some clusters, and which ones are actually outliers.

We first assign each data point $dp$ to a certain medoid $\in RS_2$ which is the nearest one to $dp$. After this step, each medoid $i \in RS_2$ is associated with a set of data points.
Proc FindKMedoids \( (D: \text{data set}, k: \text{no. of medoids}) \)

\[
\begin{align*}
\text{Begin} \\
RS &\leftarrow \emptyset; \{\text{RS is the set of medoids}\} \\
\text{Randomly pick up a data point } medoid_1 \text{ from } D; \\
RS &= RS \cup \{medoid_1\}; \\
\text{For each } dp \in D - RS \text{ do } d'(dp) = d(dp,medoid_1); \\
\text{For } i = 2 \text{ to } k \text{ do } \\
\text{Begin} & \\
\text{Find } medoid_i \in D - RS \text{ with largest } d'(medoid_i); \\
RS &= RS \cup medoid_i; \\
\text{For each } dp \in D - RS \text{ do } \\
d'(dp) &= \min\{d'(dp), d(dp,medoid_i)\} \\
\text{End; } \\
\text{return } RS \\
\text{End.}
\end{align*}
\]

Figure 5.3.2: Proc: FindKMedoids

**Initial division of the data set**

*Cluster or outlier?* Now that we get the set \( E \) of the initial division of the input data set \( X \), we check the size of each medoid-associated data subset \( \in E \). A medoid \( medoid_i \) with a small associated data subset \( E_i \) is probably an outlier, but it's not necessarily the case. Since we get the initial medoids randomly, multiple medoids are likely located in the same natural cluster, and some of them may contain very few data points, but actually they belong to a cluster instead of being an outlier. Figure 5.2.4 shows some examples of such case. From Figure 5.2.4 we can see that three medoids (\( medoid_1, medoid_2 \) and \( medoid_3 \)) are randomly selected and they form three data subsets \( (E_1, E_2 \text{ and } E_3) \). The data subset \( E_1 \) has the smallest size. However, from the figure we can easily find that \( E_1 \) is actually
a part of a natural cluster (combination of $E_1$ and $E_2$). So the size itself is not enough to
determine whether a medoid is an outlier or actually in a cluster.

![Figure 5.3.3: Ordered list of outlier subsets in the intermediate set based on $|s|/d_t(s)$](image)

Here we exploit an approach to adjust the criterion to determine whether a medoid is an
outlier or it belongs to a cluster. We first separate the data subsets $s \in E$ with some size
threshold $T_s$ which is application-oriented into set $C$ and $O$. For each subset $s$ in $O$, suppose
it is formed by medoid $medoid_i$. We not only use the size of $s$ but also use the minimum
distance between it and other medoid-associated data subsets $s \in E$ to determine the possi-
bility of $medoid_i$ being an outlier. The reason is that if $medoid_i$ forms a small data subset $s$
but it is near other data subsets, it is more likely that $medoid_i$ is a data point inside a cluster
instead of being an outlier. Practically, we use $|s|/d_t(s)$ as the criterion for the determination
(see Figure 5.3.5 for the definition of $d_t(s)$). The smaller the value of $|s|$ is and the larger
the value of $d_t(s)$ is, the smaller the value of $|s|/d_t(s)$ is and the more likely the medoid $medoid_i$
forming data subset $s$ is an outlier. It is also consistent to the case shown in Figure 5.2.4.

For the subsets in $O$, we sort them in ascending order by $|s|/d_t(s)$ ($s \in O$) into ordered list $I$. 
and find the first sharp upward point $p'(\text{see Figure 5.3.3})$. Data subsets before $p'$ in $l$ not only have small size, but also are far from other data subsets, so they can be reasonably regarded as the groups of outliers. If there is no sharp point at all in $l$, we simply set $p'$ to the one-third position. For those data subsets after $p'$ in $l$, they are more likely some data groups inside natural clusters, and we put them back into the cluster set $C$.

After the process of $\text{ClusterOrOutlier}$, it should be very unlikely the case that the size of cluster set $C$ is less than $k$ if the initial sizes $\text{RandomSize1}$ and $\text{RandomSize2}$ are large enough. If it does happen, we can just run the initial step again to make sure the size of the cluster set $C$ is at least $k$. 

**Proc DispatchDataPoints** (*RS$_2$: Set of Initial Medoids*)

Begin
  RandomSize2 ← |RS$_2$|;
  For each $i = 1, \ldots, RandomSize2$ do $E_i ← \emptyset$
  For each data point $dp$ in the current data set do
    Begin
      Find $\text{medoid}_i \in RS_2$ with smallest $d(dp, \text{medoid}_i)$;
      $E_i = E_i \cup \{dp\}$
    End;
    $E ← E_1, E_2, \ldots, E_{RandomSize2}$;
  return $E$
End.
5.3.2 Iterative phase

In the iterative phase, we first merge the initial set of clusters into $k$ clusters. In the second step, we sort clusters and outliers based on their qualities and select the worst clusters and outliers. The quality of each cluster is calculated according to the intra-relationship within clusters and the inter-relationship between clusters and outliers, and vice versa. In the third step, for clusters of the worst qualities, we exploit some methods to select the boundary data points for each of them. In the fourth step, we refine the set of clusters and outliers gradually by optimally exchanging the selected boundary data points and the worst outliers. Steps two, three and four are performed iteratively until a certain termination condition is reached. We detail each step in the following.

Merging clusters

Before we perform the cluster-outlier iterative detection process, we should first merge the current cluster set $\mathcal{C}$ to $k$ clusters. It’s an iterative process. In each iteration, two nearest clusters are found in $\mathcal{C}$ and they are merged. The distance between two clusters $C_1$ and $C_2$ is based on the diversity measurement $D_2(C_1, C_2)$ of two clusters defined in Section 5.2. The iteration step is performed until the total number of clusters in $\mathcal{C}$ is $k$. We compute the centroid of each cluster $C_i \in \mathcal{C}$ (denoted as $c_i$).
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

Sorting clusters and outliers

For each outlier \( \in O \), its nearest cluster \( \in \mathcal{C} \) is found. The distance between the cluster \( C \) and the outlier \( O \) is based on the diversity measurement \( D_1(C,O) \) defined in Section 5.2. Also its quality \( Q_o(O) \) (defined in Section 5.2) is calculated according to both the information of outliers in \( O \) and the information of clusters in \( \mathcal{C} \). Outliers with the worst qualities (Similar to the method show in Figure 5.3.3, the outliers before the first sharp point) are put into set \( O' \).

Similarly, for each cluster \( C \in \mathcal{C} \), its quality \( Q_c(C) \) (defined in section 5.2) is calculated according to not only the information of clusters in \( \mathcal{C} \), but also the information of outliers in \( O \). Clusters with the worst qualities (Similar to the method show in Figure 5.3.3, the clusters before the first sharp point) are put into set \( \mathcal{C}' \).

The worse quality a cluster has, the more likely it contains some data points which are better to be outliers. Similarly, the worse quality an outlier has, the more likely it should be included in a certain cluster.

Finding boundary data points

We call those data points in clusters which not only are farthest from the centroids of the clusters, but also have the least number of neighboring data points as boundary data points.
of the clusters. The latter factor ensures that this method does not only favor clusters of standard geometries such as hyper-spherical ones. For real-world data, the boundary data points of the clusters are ambiguous. Some boundary data points under a certain scale will be regarded as outliers under another scale, and some outliers under a certain scale are better to be treated as boundary data points under another scale. To decide which data points should be boundary data points belonging to a certain cluster and which ones should be outliers is a difficult problem. We design an algorithm to solve such a problem smoothly.

For each cluster $C_i \in C'$, we search its boundary data points based on the distance $d_{ij}$ between each data point $dp_j$ and the centroid $c_i$ of the cluster $C_i$ it belongs to, and the number of its neighbors $\varsigma(j)$. $\varsigma(j)$ is the number of data points within radius $\tau$ of $dp_j$, and $\tau$ is set proportional to $|C_i|$. We sort the data points in $C_i$ in descending order based on $d_{ij}/\varsigma(j)$ and locate the first sharp downward point $\hat{p}$. The data points before $\hat{p}$ are regarded as boundary data points of $C_i$. If the size of the boundary points of cluster $C_i$ is over a certain percentage of the cluster size, we assume that cluster $C_i$ does not have outstanding boundary data points, and we simply set its boundary data points set as $\emptyset$ (empty).

We group all boundary data points of clusters $\in C'$ into set $BDP$ and check the size of $BDP$. If its size is much larger than the size of $O'$, we should further reduce the size of $BDP$ since we don’t hope that the sizes of $BDP$ and $O'$ have too much differences to unbalance the initial division of $X$ in the consideration that the initial results of most existing algorithms
Proc ClusterOrOutlier (E: set of initial data division, $T_s$: threshold of the cluster size)

Begin

$C \leftarrow \emptyset$ \{ $C$ is the set of clusters \}

$O \leftarrow \emptyset$ \{ $O$ is the set of outliers \}

$O \leftarrow \emptyset$ \{ $O$ is the intermediate set of data subsets associated with medoids which might be outliers \}

For each set $E_i \in \text{the initial data division } E$ do

Begin

If $|E_i| \geq T_s$

$C = C \cup \{E\}$

Else $O = O \cup \{E\}$

End;

For each set $s \in O$ do

Begin

Find its nearest set $E_j \in E$;

Let $d_t(s) = D_2(s, E_j)$;

End;

Sort data subsets $\in O$ in ascending order by $\frac{|s|}{d_t(s)}$ (s is any data subset $\in O$);

Denote the ordered set list as $l$;

Find the first sharp upward point $p'$ in $l$;

$O =$ data points $\in$ data subsets before $p'$ in $l$;

$C = C \cup$ data subsets after $p'$ in $l$;

return $C$ and $O$

End.

Figure 5.3.5: Proc: ClusterOrOutlier
already contain clusters and outliers similar to the ground truth. To reduce the size of BDP, we perform another sort on the data points in BDP. Note that the previous sort of data points is for each cluster $C_i \in \mathcal{C}'$ individually, and this time the sort is on the combination of the boundary data points from all the clusters $\in \mathcal{C}'$. The criterion of the sort is also different from the previous one. This time we not only use the distance between boundary data points and the centroid of clusters they belong to and the number of their neighbors, but also integrate the information of the sizes of the clusters into the criterion. It is because clusters $\in \mathcal{C}'$ have difference sizes, and sort solely based on the distance measurement is unfair to the boundary data points of some clusters in this case. Practically, we use \(d_{ij}/(|C| \ast \varsigma(j))\) as the sorting to eliminate the effect of the size differences among clusters.

We reduce the size of BDP, making it only contain the first $|O'|$ boundary data points in the ordered boundary data point list.

**Exchanging data points in clusters and outliers**

Next step is to exchange the characteristics of outliers and boundary data points. For each outlier $O$ in $|O'|$, we add it into its nearest cluster. For each boundary data point $bdp$ in BDP, we change it into a new outlier.

The reason that we don’t exchange boundary data points *between* clusters is that the whole quality of the data division will be deteriorated if it is conducted. Following is the lemma
to support our conclusion.

**Lemma 5.3.1** Let \( d_{p1} \) be a boundary data point of cluster \( C_1 \), and \( d_{p2} \) be a boundary data point of cluster \( C_2 \). Let the new clusters be \( C^e_1 \) and \( C^e_2 \). \( Q_c(C^e_1) + Q_c(C^e_2) \leq Q_c(C_1) + Q_c(C_2) \).

**Proof sketch.** From the definition and processing of the initial division of the data set we can easily get: if \( d_{p1} \) is a boundary data point of cluster \( C_1 \), and \( C^e_1 \) is the new cluster without \( d_{p1} \) but including \( d_{p2} \), then \( Q_c(C^e_1) \leq Q_c(C_1) \). The same is for \( Q_c(C^e_2) \leq Q_c(C_2) \). So \( Q_c(C^e_1) + Q_c(C^e_2) \leq Q_c(C_1) + Q_c(C_2) \).

**Termination condition:** After the exchange step, we reorganize the united set \( U \) of the worst outlier set \( |O'\) and the reduced boundary data point set BDP as \( U = |O'| \cup BDP \). The iterative process is performed until either of the following two conditions is satisfied: the elements \( \in U \) don’t change dramatically anymore, or some threshold of the iteration number is reached.

### 5.3.3 Time and space analysis

Suppose the size of the data set is \( n \). Throughout the process, we need to keep track of the information of all points, which collectively occupies \( O(n) \) space. For the iteration step, we
**Proc MergeCluster** (C: the initial set of clusters, k: No. of Clusters)

Begin
- While |C| >
  Begin
  - Find two nearest clusters C and C in C;
  - C = C ∪ C;
  - C = C − {C};
  End;
- return C
End.

Figure 5.3.6: Proc: MergeCluster

**Proc ExchangeClusterAndOutlier** ()

Begin
- C’ is the set of clusters with worst qualities;
- O’ is the set of outliers with worst qualities;
- Get set BDP of boundary data points of all the clusters ∈ C’;
- Reduce the size of BDP if it is much larger than |O’|;
- Change the boundary data points in BDP into outliers;
- Add each outlier in O’ into its nearest cluster ∈ C;
- Calculate the quality Q(X) of the division of the data set
End.

Figure 5.3.7: Proc: ExchangeClusterAndOutlier
need space for the information of current set $\mathcal{C}$ of clusters, the current set $\mathcal{O}$ of outliers, the boundary data points of each cluster, the worst outliers and worst clusters in each iteration. The total space needed is $O(n)$. The time required for each iteration is $O(n + |\mathcal{C}| \log |\mathcal{C}| + |\mathcal{O}| \log |\mathcal{O}|)$ mainly for computation of the various of qualities and sort process. $\mathcal{C}$ and $\mathcal{O}$. So the total time required for the algorithm is $O(\mathcal{I} \ast (n + |\mathcal{C}| \log |\mathcal{C}| + |\mathcal{O}| \log |\mathcal{O}|))$ in which $\mathcal{I}$ is the threshold of the iteration number.

### 5.4 Experiments

We conducted comprehensive experiments on both synthetic and real data sets 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. Trials using data sets from real-world applications were performed comparing to other algorithms such as CURE and Shrinking.

The accuracy of a detected cluster was measured according to precision and recall. For a detected cluster $C^r_i$ and a real cluster $C^o_i$, we define the precision of $C^r_i$ with respect to $C^o_i$ as $\frac{|C^r_i \cap C^o_i|}{|C^r_i|}$ and the recall as $\frac{|C^r_i \cap C^o_i|}{|C^o_i|}$. $C^r_i$ is called a corresponding cluster of $C^o_i$ if the precision and recall of $C^r_i$ with respect to $C^o_i$ are high.

Some data sets which have natural clusters and outliers will converge to form stable sets of clusters and outliers, others which either do not have natural clusters and outliers at
all, or are not under proper parameters, do not converge at all. In our experiments, we empirically found that after at most 30 iterations, the movements of most of the data sets which converge eventually will be stabilized, so we set the maximum iteration number $\mathcal{I}$ as 30.

Other parameters are optimized as follows. For each parameter, we did experiments to check its values which are best for the runtime of the algorithm or the performance in terms of the resulted clusters and outliers. Values which are best for runtime usually are not best for performance though. So we shall balance those two best values to get the value of this parameter.

Here we briefly present the procedure of determining the value of $\text{Const1}$ which is used to determine the RandomSize1 of the random sample. We did experiments with $\text{Const1}$ of various values, on different data sets. Figure 5.4.1 shows the performance (the average of precisions and recalls of the resulting clusters) and runtime of our algorithm on Wine data. From the figure we can see that $\text{Const1}$ values best for runtime may not be best
for performance. We balance those two values on different data sets, and set the value of Const1 as 4 for Wine data.

5.4.1 Experiments on high-dimensional data sets

To test the scalability of our algorithm over dimensionality and data size, we designed a synthetic data generator to produce data sets with clusters. The sizes of the data sets vary from 5,000, 10,000, ... to 100,000, and the dimensions of the data sets vary from 10, 20, ... to 60. Each data set contains five clusters, with the points in each cluster generated in normalized distributions. An additional 5% of data points are added randomly to each data set as noise. Clusters and outliers were effectively detected by our algorithm in all tests performed on these high-dimensional data sets. Figure 5.4.2 shows the running time of 20 groups of data sets with dimensions increasing from 10 to 60. Each group has a fixed data size (the noisy data points are not counted). Figure 5.4.3 shows the running time of 11 groups of data sets with sizes increasing from 5,000 to 100,000. Each of these 11 groups has fixed number of dimensions. The two figures indicate that our algorithm is scalable over dimensionality and data size.
5.4.2 Data sets

The real data sets were obtained from UCI Machine Learning Repository [10]. The first one is the Wine Recognition data (simplified as Wine data). It 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. The data set has three clusters with the sizes of 59, 71 and 48.

The second data set is Ecoli data. It contains data regarding Protein Localization Sites. This data set is made up of 336 instances, with each instance having seven features. It contains 8 clusters with the sizes of 143, 77, 52, 35, 20, 5, 2 and 2.

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 $i = 1, 2, ..., 10$.

Due to the space limitation, we show the comparison of the testing results of our algorithm on Wine and Ecoli data with those of other algorithms, and demonstrate how COID can improve the performance of other algorithms on Wine and Pendigits data.

5.4.3 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 [35].

**Shrinking:** We also conducted experiments using Shrinking algorithm [90] on real data sets. Shrinking is a data preprocessing technique which optimizes the inner structure of data inspired by the Newton’s Universal Law of Gravitation [70] in the real world. Shrinking-based multi-dimensional data analysis approach first moves data points along the direction of the density gradient, thus generating condensed, widely-separated clusters. It then detects clusters by finding the connected components of dense cells. Then it uses a cluster-wise evaluation measurement to compare the clusters from different cases and select the best clustering results.

**COID:** Our clustering and outlier detection version is based on the algorithm described in the previous sections. First suitable medoids for natural clusters are searched and the initial clusters and outliers are detected, then clusters and outliers are iteratively modified and refined according to the relationship among them.

First experiments were conducted on Wine data. We applied CURE algorithm on the Wine
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

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 \( \alpha \) and the number of representative points r with different values
and found the best clustering result among those parameter values. Due to space limitation,
we just present one of the best results of CURE. Table 5.4.1 shows the clustering results of
CURE algorithm when \( \alpha \) is equal to 0.3, and r is 10.

| \( i \) | \( |C^o_i| \) | \( |C^s_i| \) | \( |C^o_i \cap C^s_i| \) | precision(%) | recall(%) |
|-------|---------|---------|----------------|-------------|-----------|
| 1     | 59      | 72      | 54             | 75.00       | 91.52     |
| 2     | 71      | 50      | 41             | 82.00       | 57.77     |
| 3     | 48      | 46      | 26             | 56.52       | 54.16     |

Table 5.4.1: Clustering result of CURE for Wine data as \( \alpha = 0.3 \) and r=10

We then applied Shrinking-based clustering algorithm on Wine data. Table 5.4.2 shows the
clustering results of the Shrinking algorithm.

| \( i \) | \( |C^o_i| \) | \( |C^s_i| \) | \( |C^o_i \cap C^s_i| \) | precision(%) | recall(%) |
|-------|---------|---------|----------------|-------------|-----------|
| 1     | 59      | 53      | 53             | 100         | 89.83     |
| 2     | 71      | 52      | 51             | 98.08       | 71.83     |
| 3     | 48      | 46      | 43             | 93.48       | 89.58     |

Table 5.4.2: Clustering results of Shrinking algorithm for Wine data

Our COID algorithm is also performed on the Wine data. Table 5.4.3 shows the clustering
results of our algorithm. We can see the overall clustering result of COID algorithm is
better than CURE. As for Shrinking, although the average precision of the testing result of
COID is lower than that of Shrinking (93.83% vs 97.05%), the average recall of the testing
result of COID is much higher than that of Shrinking (91.71% vs 83.75%).

\[
\begin{array}{ccc}
  & i=1 & i=2 & i=3 \\
| C^0_i | & 59 & 71 & 48 \\
| C^0_i | & 56 & 73 & 46 \\
| C^0_i \cap C^0_i | & 53 & 65 & 45 \\
precision(\%) & 94.64 & 89.04 & 97.82 \\
recall(\%) & 89.83 & 91.54 & 93.75 \\
\end{array}
\]

Table 5.4.3: Clustering results of COID algorithm for Wine data

Experiments are performed also on Ecoli data. 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, so we set the cluster number parameter k to 5, set the shrinking factor \( \alpha \) and the number of representative points r with various of values and select the best clustering result. Table 5.4.4 shows the clustering results of CURE algorithm when \( \alpha \) is equal to 0.8, and r is 20.

\[
\begin{array}{ccccccccc}
  & i=1 & i=2 & i=3 & i=4 & i=5 & i=6 & i=7 & i=8 \\
| C^0_i | & 143 & 77 & 52 & 35 & 20 & 5 & 2 & 2 \\
| C^0_i | & 120 & 67 & 32 & N/A & N/A & 3 & N/A & N/A \\
| C^0_i \cap C^0_i | & 115 & 41 & 30 & N/A & N/A & 3 & N/A & N/A \\
precision(\%) & 95.83 & 61.19 & 93.75 & N/A & N/A & 100 & N/A & N/A \\
recall(\%) & 80.41 & 53.24 & 57.69 & N/A & N/A & 60.00 & N/A & N/A \\
\end{array}
\]

Table 5.4.4: Clustering result of CURE for Ecoli data as \( \alpha=0.8 \) and r=20

We then applied Shrinking-based clustering algorithm on Ecoli data. Table 5.4.5 shows the clustering results of the Shrinking algorithm.

Our COID algorithm is also performed on the Ecoli data. Table 5.4.6 shows the clustering
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

results of our algorithm. Compared to the clustering results of CURE and Shrinking algorithms, COID has the most information of the first 3 largest natural clusters among these algorithms.

Table 5.4.5: Clustering result of Shrinking algorithm for Ecoli data

<table>
<thead>
<tr>
<th></th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
<th>i=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0^i$</td>
<td>143</td>
<td>77</td>
<td>52</td>
<td>35</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$C_1^i$</td>
<td>135</td>
<td>22</td>
<td>68</td>
<td>49</td>
<td>11</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$C_1^i \cap C_0^i$</td>
<td>130</td>
<td>22</td>
<td>43</td>
<td>32</td>
<td>10</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>precision(%)</td>
<td>96.30</td>
<td>100</td>
<td>63.24</td>
<td>65.31</td>
<td>90.91</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>recall(%)</td>
<td>90.91</td>
<td>28.57</td>
<td>82.69</td>
<td>91.43</td>
<td>50.00</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.4.6: Clustering result of COID algorithm for Ecoli data

<table>
<thead>
<tr>
<th></th>
<th>i=1</th>
<th>i=2</th>
<th>i=3</th>
<th>i=4</th>
<th>i=5</th>
<th>i=6</th>
<th>i=7</th>
<th>i=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0^i$</td>
<td>143</td>
<td>77</td>
<td>52</td>
<td>35</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$C_1^i$</td>
<td>137</td>
<td>89</td>
<td>70</td>
<td>N/A</td>
<td>N/A</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>$C_1^i \cap C_0^i$</td>
<td>130</td>
<td>56</td>
<td>47</td>
<td>N/A</td>
<td>N/A</td>
<td>4</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>precision(%)</td>
<td>94.89</td>
<td>62.92</td>
<td>67.14</td>
<td>N/A</td>
<td>N/A</td>
<td>100</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>recall(%)</td>
<td>90.90</td>
<td>72.72</td>
<td>90.38</td>
<td>N/A</td>
<td>N/A</td>
<td>80.00</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 5.4.7: Iteration of COID algorithm on clustering result of CURE algorithm on Wine data

<table>
<thead>
<tr>
<th>i</th>
<th>$O'$</th>
<th>BDP</th>
<th>match</th>
<th>average precision(%)</th>
<th>average recall(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>71.17</td>
<td>67.81</td>
</tr>
<tr>
<td>1</td>
<td>70.36</td>
<td>110</td>
<td>66</td>
<td>71.24</td>
<td>67.90</td>
</tr>
<tr>
<td>2</td>
<td>44</td>
<td>19</td>
<td>74</td>
<td>98 109 124</td>
<td>71.91</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>129</td>
<td>62</td>
<td>59</td>
<td>3</td>
<td>82.96</td>
</tr>
<tr>
<td>21</td>
<td>124</td>
<td>110</td>
<td>62</td>
<td>2</td>
<td>83.44</td>
</tr>
<tr>
<td>22</td>
<td>62</td>
<td>43</td>
<td>110</td>
<td>4</td>
<td>83.25</td>
</tr>
</tbody>
</table>
CHAPTER 5. A CLUSTER-OUTLIER ITERATIVE DETECTION APPROACH

5.4.4 Using COID to improve other algorithms

We also did experiments to test how COID algorithm can improve the clustering and outlier detection results of other algorithms. Here we demonstrate how COID can improve the testing result of Wine data and Pendigits data from CURE algorithm.

Table 5.4.1 in the previous subsection shows the clustering result of CURE for Wine data. There are three clusters detected by CURE algorithm, with the size of 72, 50 and 46. Actually there are also 10 other data points in Wine data which are not included in any of these three clusters, and we regard them as the initial outliers. Since we already have the initial clusters and outliers in this case, we do not need to do the initialization phase of COID algorithm, and we can also skip the MergeCluster subprocess in the iterative phase.

Table 5.4.7 shows the change of the status of the testing result of Wine data. In the Table

<table>
<thead>
<tr>
<th>i</th>
<th>D′</th>
<th>BDP</th>
<th>match</th>
<th>average precision (%)</th>
<th>average recall (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>79.22</td>
<td>49.69</td>
</tr>
<tr>
<td>1</td>
<td>3151 3994</td>
<td>3978 499</td>
<td>0</td>
<td>79.71</td>
<td>49.70</td>
</tr>
<tr>
<td>2</td>
<td>34 40 40 28 87</td>
<td>3994 3151 3318</td>
<td>2</td>
<td>79.71</td>
<td>49.70</td>
</tr>
<tr>
<td>3</td>
<td>3151 3994</td>
<td>2887 6838</td>
<td>3</td>
<td>78.78</td>
<td>49.69</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>23</td>
<td>3686 1216 824 10111 10718</td>
<td>10492 8387 3686 8891 3643</td>
<td>8</td>
<td>81.58</td>
<td>49.83</td>
</tr>
<tr>
<td>24</td>
<td>10111 10718 247 1482</td>
<td>10492 8387 3686 8891</td>
<td>8</td>
<td>81.93</td>
<td>49.82</td>
</tr>
<tr>
<td>25</td>
<td>10492 3686 499</td>
<td>247 1482 10111</td>
<td>10718 8643 8387</td>
<td>10</td>
<td>82.33</td>
</tr>
</tbody>
</table>

Table 5.4.8: Iteration of COID algorithm on clustering result of CURE algorithm on Pendigits data
5.4.7, the column $O'$ contains ids of those outliers with the worst qualities, although there might be other outliers at that moment. Similarly, the column BDP presents ids of those boundary data points of clusters which have worst qualities. The contents of these two columns are to be exchanged with each other. They constitute the set $U$ ($U = O' \cup BDP$).

The column *match* shows how many elements in $U$ in current iteration are similar to those in the previous iteration. The average precision and the average recall are according to the clustering results.

The first line lists the original testing result from the CURE algorithm on Wine data, so the worst outliers, worst boundary data points and the match columns are not available yet. The average precision is 71.17% (resulting from the average of the precisions of the clustering result from table 5.4.1: 75.00%, 82.00% and 56.52%). The average recall is 67.81% (resulting from the average of the recalls of the clustering result from table 5.4.1: 91.52%, 57.77% and 54.16%).

In each iteration, the clusters and outliers are adjusted and modified according to their intra-relationship and inter-relationship. Outliers with the worst qualities are selected, so are those "worst" boundary data points. From table 5.4.7 we can see that the average precision and average recall of the clustering result improve as the iteration goes on. The iterations are executed till iteration 22, when the match of the set $U$ to that of the previous iteration 21 is 100%. The updated average precision is 83.25% and the updated average
recall is 79.42%, both much higher than the original ones.

We tested how COID could improve the performance of CURE algorithm on Pendigits data as well which contains 10992 instances and has 16 attributes. Table 5.4.8 shows the change of the status of the testing result of Pendigits data. The average precision and average recall of the clustering result also improve as the iteration goes on.

5.5 Conclusion and discussion

In this chapter, we presented a novel cluster and outlier iterative detection approach. The method can effectively and efficiently improve the qualities of clusters as well as outliers in a noisy data set of multi-dimensions. Clusters are detected and adjusted according to the intra-relationship within clusters and the inter-relationship between clusters and outliers, and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached. Besides treating the clusters and outliers as concepts of the same importance, we also hope to smooth the problem of the lack of match between the ground truth of the real data sets and their available features.
Chapter 6

Dynamic Clustering and Indexing of Multi-Dimensional Data

Large volumes of data with high dimensionality are being generated in many fields. Most existing indexing techniques degrade rapidly when dimensionality goes higher. ClusterTree is a new indexing approach representing clusters generated by any existing clustering approach. It is a hierarchy of clusters and subclusters which incorporates the cluster representation into the index structure to achieve effective and efficient retrieval. In this chapter, we propose a new ClusterTree+ indexing structure from the conceptual and theoretical point of view, which has new features from the time perspective. Each new data item is added to the ClusterTree with the time information which can be used later in the data update
process for the acquisition of the new cluster structure. This approach guarantees that the ClusterTree is always in the most updated status which can further promote the efficiency and effectiveness of data insertion, query and update. This approach is highly adaptive to any kind of clusters.

6.1 Introduction

Recently large volumes of data with high dimensionality are being generated in many fields. Many approaches have been proposed to index high-dimensional data sets for efficient querying. Existing multidimensional tree-like indexing approaches can be further classified into two categories: space partitioning and data partitioning. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data set efficiently since it may greatly increase the amount of data accessed for a query.

Among those approaches, the ClusterTree [92] is the first work towards building efficient index structure from clustering for high-dimensional data sets to facilitate efficient queries. The ClusterTree has the advantage of supporting efficient data query for high-dimensional data sets. A ClusterTree is a hierarchical representation of the clusters of a data set. It
organizes the data based on their cluster information from coarse level to fine, providing an efficient index structure on the data according to clustering. Like many other structures, it has two kinds of nodes: internal and leaf nodes. The internal nodes include pointers to subclusters of this cluster, the bounding sphere for each subcluster and the number of data points in each subclusters. The leaf nodes contain pointers to the data points. For each cluster, the ClusterTree calculates the following parameters: the number of data points, the centroid \( c = (c_1, c_2, \ldots, c_d) \) can be calculated by:

\[
(6.1.1) \quad c_i = \frac{\sum_{j=1}^{N} o_{ji}}{N}, \quad 1 \leq i \leq d,
\]

where \( N \) is the number of the data points in the cluster and \( o_{ji} \) is the \( i \)-th value of data point \( o_j \) in the cluster. Thus, each cluster is represented by a hyper-sphere \( S \). So there may be some empty regions which contain no data, and two bounding hyper-spheres of two different clusters may overlap. We can define the density of the cluster as:

\[
(6.1.2) \quad \text{Density}_c = \frac{\text{number of points in } C}{\text{volume of } S} = \frac{\text{number of points in } C}{\frac{2\pi^{d/2}r^d}{d\Gamma\left(\frac{d}{2}\right)}}.
\]

The gamma function \( \Gamma(x) \) is defined as:

\[
(6.1.3) \quad \Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt,
\]
where \( \Gamma(x + 1) = \lambda \Gamma(x) \) and \( \Gamma(1) = 1 \). When the density of a cluster falls below a pre-selected threshold or the number of the data points in the cluster is larger than a pre-selected threshold, the cluster will be decomposed into several smaller clusters (subclusters).

Nowadays a large proportion of the data sets is time related, and the existence of the obsolete data in the data sets may seriously degrade the data set processing. However, few approaches are proposed to implement the maintenance of the whole data set to get rid of the obsolete data and keep the data set always in the most updated status for the convenience and effectiveness of data set process such as query and insertion.

In this chapter, we propose a new ClusterTree\(^+\) indexing structure from the conceptual and theoretical point of view, which has new features from the time perspective. Each new data item is added to the ClusterTree\(^+\) with the time information which can be used later in the data update process for the acquisition of the new cluster structure. This approach guarantees that the ClusterTree\(^+\) is always in the most updated status which can further promote the efficiency and effectiveness of data insertion, query and update. Our approach is highly adaptive to any kind of clusters.

The rest of the chapter is organized as follows. Section 2 summaries the related work on index structure design including the ClusterTree. Section 3 introduces our modified ClusterTree approach – ClusterTree\(^+\). Section 4 presents the processing of ClusterTree\(^+\). Section 5 gives the conclusion.
6.2 ClusterTree$^+$ and its Construction

A large proportion of a data set may be time related, and the existence of the obsolete data in the data set may seriously degrade the data set processing. However, few approaches are proposed to implement the maintenance of the whole data set to get rid of the obsolete data and keep the data set always in the most updated status for the convenience and effectiveness of data set process such as query and insertion. Although ClusterTree can efficiently handle the processing of data sets with high dimensionality, it doesn’t consider the dynamic data update issue.

Here we present a modified version of ClusterTree: ClusterTree$^+$, which is based on the design of the ClusterTree and enhance its capability of handling dynamic data insertions, queries and deletions.
There can be several different ways to associate the time information with the original ClusterTree structure. The typical three approaches are as follows:

- Directly add time information into the ClusterTree as another dimension.
- Use a simple queue to process the time issue.
- Use an individual B+ tree-similar structure to handle the time information.

The advantage of the first approach is that it simplifies the implementation. We can slightly adjust the original algorithm to support the queries which may include the time information and deletions based on the time period which is specified by users. The disadvantage is that if we directly take time information as an extra dimension of the data set, the clustering results will significantly change. Two data points which are very close to each other in the original space may not even exist in the same cluster after adding the time dimension, since the times the data are inserted into the indexing structure may be quite distant to each other. So it will dramatically degrade the result of the queries which are simply based on the data itself without considering the time issue.

The idea of using a simple queue to process the time issue is straightforward. But as a linear structure, the efficiency is the major problem.

Using an individual B+ tree [71]-similar structure to handle the time information can support both time-related queries and time-irrelevant queries. For time-irrelevant queries, we
can use algorithms similar to those of the original ClusterTree. As for time-related queries including range queries and $p$-Nearest Neighbors queries, we can use the intersection of the searching result of both modified ClusterTree structure and $B^+$ tree-similar structure to get the final result. This approach can also efficiently support user-specified periodic deletions to get rid of the obsolete data in the data set.

So we choose the last kind of approach to set up the ClusterTree$^+$ indexing structure as our solution to solve the data update problem of high dimensional data sets. The ClusterTree$^+$ has two separate structures, one is a modified ClusterTree structure – ClusterTree$^*$, the other is a modified $B^+$ tree – $B'$ tree.

The ClusterTree$^*$ is a hierarchical representation of the clusters of a data set. It has two kinds of nodes: internal and leaf nodes. The internal nodes are defined as:

$$\text{Node} : \text{Node}_id, \gamma, \text{ot}, \text{nt}, (\text{Entry}_1, \text{Entry}_2, \ldots, \text{Entry}_\gamma)$$

$$\left( m_{\text{node}} \leq \gamma \leq M_{\text{node}} \right)$$

$$\text{Entry}_i : (SC_i, BS_i, SN_i),$$

where $\text{Node}_id$ is the node identifier, $\gamma$ is the number of the entries in the node, $\text{ot}$ is the oldest time when data are inserted into that node or its descendants, $\text{nt}$ is the newest time when data are inserted into that node or its descendants, and $m_{\text{node}}$ and $M_{\text{node}}$ define the minimum and maximum numbers of entries in the node. An entry is created for each subcluster of the cluster for which the current non-leaf node represents. In entry $\text{Entry}_i$, $SC_i$ is a pointer to the $i$-th subcluster, $BS_i$ is the bounding sphere for the subcluster and $SN_i$
is the number of data points in the $i$-th subcluster.

The leaf nodes are defined as:

$$\text{Leaf} : \text{Leaf.id}, \gamma, o.t., n.t., (\text{Entry}_1, \text{Entry}_2, \ldots, \text{Entry}_\gamma)$$

(6.2.2)

$$(m_{leaf} \leq \gamma \leq M_{leaf}),$$

$$\text{Entry}_i : (\text{ADR}_i, T_i, L_i),$$

where $\gamma$ is the number of data points contained in the leaf node, and $m_{leaf}$ and $M_{leaf}$ are the minimum and maximum numbers of entries. $\text{Entry}_i$ contains the address of the data point residing at the secondary storage ($\text{ADR}_i$), the time information when the data point is inserted into the structure ($T_i$), and the link to the time data point in the $B'$ tree ($L_i$).

The $B'$ tree indexes on the time data which corresponds to the times the data were inserted into the structure. It originates from the $B^+$ tree with some modifications as follows:

- There is no minimum number requirement of entries in internal and leaf nodes so that there will be no cases of underflow. This corresponds to the nature of the $B'$ tree that the time data in it will be deleted in batch for the user-specified deletion requirement.

- In the leaf nodes, each entry has an extra field which is a link to the data point it is associated with in the ClusterTree*. In this case we can traverse from the $B'$ tree back to the ClusterTree* efficiently.

An example of the ClusterTree$^+$ structure is shown in Figure 1. It shows that currently there are 9 data which are inserted at time 1, 2, 3, 4, 5, 6, 7, 9 and 10. The root node of
the ClusterTree\textsuperscript{*} has two entries. The first entry also has two entries. At the third level, the first leaf node has three data entries which are inserted at time 1, 3 and 6. The $L_t$ fields of these entries of the leaf node are connected with the leaf nodes of the $B'$ tree which is on the bottom-right part of the Figure 1.

### 6.2.1 Construction

The construction of ClusterTree\textsuperscript{+} includes the construction of ClusterTree\textsuperscript{*} and the construction of $B'$ tree in parallel. The construction of the ClusterTree\textsuperscript{*} is similar to the construction of ClusterTree except that each internal node and leaf node should set the $nt$ (newest time) and $ot$ (oldest time) as the current time. This is because we don’t know the insertion times of the original data points in the data set, so we have to set the insertion times of all the original data points as the current one. Later when new data points are inserted into the structure, different times can be recorded into the structure. Meanwhile, we create a leaf node in $B'$ tree which includes the current time data. All $L$ field of the entries in the leaf node of ClusterTree\textsuperscript{*} should be pointed to the created leaf node in $B'$ tree.

### 6.3 Processing of the ClusterTree\textsuperscript{+}

There are three major processing of the ClusterTree\textsuperscript{+}: insertion, query and deletion.
6.3.1 Insertion

For a new coming data point, we classify it into one of three categories:

- **Cluster points**: are either the duplicates of or very close to some data points in a cluster within a given threshold.

- **Close-by points**: are the data points which are neighbors to some points in the clusters within a given threshold.

- **Random points**: are the data points which are either far away from all of the clusters and can not be bounded by any bounding sphere of the ClusterTree, or might be included in the bounding spheres of the clusters at each level, but they do not have any neighboring cluster points within a given threshold.

Thus depending on the type of the new coming data point, we can apply the insertion algorithm of ClusterTree to recursively insert data point to a certain leaf node of ClusterTree\(^*\) accompanied by the insertion time information in the T(time) field of the new entry in that leaf node, while inserting a new entry which includes the insertion time info into the \(B'\) tree. We then point the L(link) field of the new entry in the certain leaf node in ClusterTree\(^*\) to the new entry in the leaf node of \(B'\) tree, and point the L(link) field of the new entry in the certain leaf node in \(B'\) tree to the new entry in the leaf node of ClusterTree\(^*\).
6.3.2 Query

There are two kinds of queries, one is the time-irrelevant queries which include the range queries and \( \text{P} \) Nearest Neighbor queries, and the other is the time-related queries which include certain time period requirement. For example, some users may ask for the neighbors to a certain data point which are inserted into the structure in almost the same time as the insertion time of that data point.

We can depend on the original ClusterTree query algorithm to solve the former kind of queries. As for the latter one, we can solve it as follows:

**Algorithm: time-related queries**

*Input:* a data point, a time stamp;

*Output:* the set of data points in ClusterTree\(^*\) which satisfy the query requirement;

*Algorithm steps:*

- **find** the set \( a \) of candidate data points in the ClusterTree\(^*\);

- at the same time **search** the entry \( x \) in \( B^t \) tree whose time data is closest to the query time stamp;

- **find** the set of the entries in \( B^t \) tree which are in a certain threshold in time distance to the entry \( x \);
• find the set $b$ of corresponding data points in $\text{ClusterTree}^*$ using the $L$ field in the entries in $B'$ tree;
• get the result set of data sets by the intersection of set $a$ and $b$.

6.3.3 Deletion

In many systems the obsolete data should be deleted periodically. Managers may want to delete those data which are inserted into the system a certain time ago, or they want to delete those data inserted during a certain period. Also they can simply indicate to the data system to automatically adjust itself. The $\text{ClusterTree}^+$ can support such user-specified deletions.

Algorithm: time-related deletion

Input: a time stamp $ts$;

Output: the new $\text{ClusterTree}^+$ which has got rid of the obsolete data (the data inserted before the specified time stamp);

Algorithm steps:

• find the entry $x$ in $B'$ tree whose time data is left-closest to the time stamp $ts$ ("left-closest" means that it is the newest one which is older than the specified time stamp $ts$);
• get the set $a$ of entries in the $B'$ tree which are older than the entry $x$;
• **search** for the set $b$ of corresponding data points in $\text{ClusterTree}^*$ using the $L$ field in the entries in $B^d$ tree;

• **cut** those entries in the $B^d$ tree recursively which are older than entry $x$;

• **cut** set $a$ in the $B^d$ tree;

• **cut** set $b$ in the $\text{ClusterTree}^*$.

**Algorithm: time-related deletion**

*Input:* time stamp $ts_1$, time stamp $ts_2 (ts_1 \leq ts_2)$;

*Output:* the new $\text{ClusterTree}^+$ which has got rid of the data inserted during the period between specified time stamps $ts_1$ and $ts_2$;

*Algorithm steps:*

• **find** the entry $x$ in $B^d$ tree whose time data is right-closest to the time stamp $ts_1$ (“right-closest” means that it is the oldest one which is newer than the specified time stamp);

• **find** the entry $y$ in $B^d$ tree whose time data is left-closest to the time stamp $ts_2$;

• if the time data of entry $x$ is newer than that of entry $y$, **exit**;

• **get** the set $a$ of entries in the $B^d$ tree which are between entry $x$ and $y$;

• **find** the set $b$ of corresponding data points in $\text{ClusterTree}^*$ using the $L$ field in the entries in $B^d$ tree;
• **cut** those entries in the $B^t$ tree recursively which are antecedents of set $a$;

• **cut** set $a$ in the $B^t$ tree;

• **cut** set $b$ in the ClusterTree$^*$.  

**Algorithm: automatic adjustment**

*Output:* the new ClusterTree$^+$ which has been automatically adjusted;

*Algorithm steps:* recursively check each subcluster:

• if the gap between the subcluster’s nt filed (new time inserted) is over certain threshold, delete the whole subcluster or move it to a secondary memory because it means that it contains no new data; Also delete the corresponding entries in the $B^t$ tree;

• if the old data density of a subcluster is over certain threshold, then subcluster should be reorganized in order to get rid of the old part;

• if there are two subclusters which are “close” enough to each other and their time “nature” are similar enough to each other, they can be merged into one subcluster for a more compact and reasonable vision.
6.4 Conclusion

In this chapter, we have proposed a modified version of ClusterTree – ClusterTree+ from the conceptual and theoretical point of view, which can efficiently support the time-related queries and user-specified deletions. The ClusterTree+ can keep the data set always in the most updated status to promote the efficiency and effectiveness of data insertion, query and update. Further experiments will be available to support the analysis of the ClusterTree+.

This approach can be helpful in the fields of data fusion where the data evolve dynamically and regular approaches often fail to solve the problem of keeping a certain system always containing the most updated data. This approach can dynamically supervise the data status of the system and efficiently get rid of obsolete data, and at the same time, reorganize the structure of the data set.
Chapter 7

Conclusion and Future Work

This chapter concludes our previous work and discuss the future work. In the previous chapters, we have studied the following four problems: (1) shrinking-based data preprocessing approach and shrinking-based clustering algorithm; (2) shrinking-based dimension reduction approach; (3) iteratively detecting clusters and outliers based on their inter-relationship, and on the intra-relationship within clusters, and within outliers, respectively; (4) indexing time-related multi-dimensional data sets.
CHAPTER 7. CONCLUSION AND FUTURE WORK

In the future, we will extend our work along the following directions: (1) cluster and outlier detection design in subspace; (2) dynamically insertion for indexing structure of time-related multi-dimensional data; and (3) combination of fuzzy clustering and shrinking-based data analysis approaches.

7.1 Conclusion

The fast-growing, tremendous amount of data has far exceeded our human ability for comprehension without powerful tools. It is really important to design the tools to extract the valuable knowledge embedded in the vast amounts of data. This dissertation focuses on effective and efficient mining of novel, interesting and significant patterns from real data sets. To be specific, we study the following four problems: (1) shrinking-based data preprocessing approach and shrinking-based clustering algorithm; (2) shrinking-based dimension reduction approach; (3) iteratively detecting clusters and outliers based on their inter-relationship, and on the intra-relationship within clusters, and within outliers, respectively; (4) indexing time-related multi-dimensional data sets.
7.1.1 Shrinking-based data preprocessing and clustering algorithms

Real-world data tend to be dirty, incomplete, and inconsistent. Data preprocessing can greatly benefit the utilization and exploration of real data. It can improve the quality of the data, thereby helping to improve the accuracy and efficiency of the subsequent mining process. There are various data preprocessing techniques such as data cleaning, data integration, data transformations, data reduction, etc.

In chapter 3, we propose a novel data preprocessing technique called shrinking which optimizes the inherent characteristic of distribution of data. 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 (and evaluated by their compactness). 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.
We conduct an extensive study on both synthetic data sets and real-world data sets: Wine data, Ecoli data and Pendigits data [10]. We compare the testing results of our algorithms with other existing algorithms such as CURE, OPTICS, Birch. The experimental study has shown that our algorithms are effective and efficient comparing to other algorithms. Also our algorithms are robust to the parameters and scalable to large data sets. We also demonstrate how the shrinking preprocessing will solely improve the performance of well known clustering algorithms like OPTICS, CURE, and Birch.

7.1.2 Shrinking-based dimension reduction algorithms

It is well acknowledged that in the real world a large proportion of data has irrelevant features which may cause a reduction in the accuracy of some algorithms. High dimensional data sets continue to pose a challenge to clustering algorithms at a very fundamental level. One of the well known techniques for improving the data analysis performance is the method of dimension reduction in which data is transformed to a lower dimensional space while preserving the major information it carries, so that further processing can be simplified without compromising the quality of the final results. Dimension reduction is often used in clustering, classification, and many other machine learning and data mining applications.

In chapter 4 we present continuous research on data analysis based on our previous work on
the shrinking approach. Following our previous work on the shrinking method for multi-dimensional data analysis in full data space, we propose a shrinking-based dimension reduction approach which tends to address the inadequacies of current clustering algorithms in handling multi-dimensional data, and to solve the dimension reduction problem from a new perspective. In this approach data are moved along the direction of the density gradient, thus making the inner structure of data more prominent. It is conducted on a sequence of grids with different cell sizes. Dimension reduction process is performed based on the difference of the data distribution projected on each dimension before and after the data-shrinking process. Those dimensions with dramatic variation of data distribution through the data-shrinking process are selected as good dimension candidates for further data analysis.

This approach can assist to improve the performance of existing data analysis approaches. We demonstrate how this shrinking-based dimension reduction approach affects the clustering results of well known algorithms. We conducted comprehensive experiments to assess the accuracy and efficiency of the proposed approach. We tested our approach on three data sets from real applications and demonstrate that it will help improve the performance of existing clustering algorithms such as OPTICS and BIRCH as well as data visualization tools such as VizCluster[57].
7.1.3 Cluster and Outlier Iterative Detection

Nowadays many data mining algorithms focus on clustering methods. There are also a lot of approaches designed for outlier detection. We observe that, in many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise. Thus, it is necessary to treat clusters and outliers as concepts of the same importance in data analysis.

In chapter 5 we present COID, a cluster-outlier iterative detection algorithm, tending to detect the clusters and outliers in another perspective for noisy data sets. In this algorithm, clusters are detected and adjusted according to the intra-relationship within clusters and the inter-relationship between clusters and outliers, and vice versa. The adjustment and modification of the clusters and outliers are performed iteratively until a certain termination condition is reached. We explore a new way to define the distance between two clusters, the distance between two outliers, and the one between a cluster and an outlier. We apply some novel formula to define the qualities of the clusters, the outliers, and the division of the whole data set.

We tested our algorithm using both synthetic and real-world data. We demonstrate step by step how the outliers and the boundary points of some clusters exchanged to gain better qualities of both outliers and clusters. We also test the scalability of our algorithm over
dimensionality and data size. We compare our algorithm with other existing approaches like CURE and demonstrate that our algorithm outperform others, and we also show how COID can help improve the performance of other algorithms like CURE.

7.1.4 indexing time-related multi-dimensional data sets

Large volumes of data with high dimensionality are being generated in many fields. Although most of them can efficiently support nearest neighbor search for low dimensional data sets, they degrade rapidly when dimensionality goes higher. Also the dynamic insertion of new data can cause original structures no longer handle the data set efficiently since it may greatly increase the amount of data accessed for a query. However few approaches are proposed to implement the maintenance of the whole data set to get rid of the obsolete data and keep the data set always in the most updated status for the convenience and effectiveness of data set process such as query and insertion. ClusterTree is a new indexing approach representing clusters generated by any existing clustering approach. It is a hierarchy of clusters and subclusters which incorporates the cluster representation into the index structure to achieve effective and efficient retrieval.

In chapter 6 we propose a new ClusterTree+ indexing structure from the conceptual and theoretical point of view, which has new features from the time perspective. Each new data item is added to the ClusterTree with the time information which can be used later in the
data update process for the acquisition of the new cluster structure. This approach guarantees that the ClusterTree is always in the most updated status which can further promote the efficiency and effectiveness of data insertion, query and update. We use an individual $B^+$ tree [71]-similar structure to handle the time information which can support both time-related queries and time-irrelevant queries. For time-irrelevant queries, we use algorithms similar to those of the original ClusterTree. As for time-related queries including range queries and $p$-Nearest Neighbors queries, we use the intersection of the searching result of both modified ClusterTree structure and $B^+$ tree-similar structure to get the final result. This approach can also efficiently support user-specified periodic deletions to get rid of the obsolete data in the data set, and it is highly adaptive to any kind of clusters.

7.2 Future Work

We have studied four interesting problems in clustering, outlier detection and indexing multi-dimensional data. In the future, we will extend our work along the following directions: (1) cluster and outlier detection design in subspace; (2) dynamically insertion for indexing structure of time-related multi-dimensional data; and (3) combination of fuzzy clustering and shrinking-based data analysis approaches.
7.2.1 Cluster and outlier iterative detection in subspace

In chapter 5, we observe that, in many situations, clusters and outliers are concepts whose meanings are inseparable to each other, especially for those data sets with noise, and proposed COID, a cluster-outlier iterative detection approach. However, clustering and outlier detection approaches are not always efficient and effective when applied in full data space. In chapter 4, we mentioned that in high dimensional spaces, for a given cluster, not all dimensions may be relevant to it. Data with irrelevant features may cause a reduction in the accuracy of some algorithms. We would like to explore a new approach $subCOID$, tending to explore cluster-outlier iterative detection approaches in subspace. In our approach, each cluster is associated with its own subset of dimensions, so is each outlier. We first find some initial (rough) sets of clusters and outliers. Based on the initial sets, we gradually improve the clustering and outlier detection results. In each iteration, the partition, subsets of dimensions and compactness of each cluster are modified and adjusted based on intra-relationship among clusters and the inter-relationship between clusters and outliers. The subset of dimensions and quality rank of each outlier is associated with is modified and adjusted based on relationship among outliers and the inter-relationship between clusters and outliers.

PROCLUS tends to solve the similar problem, but it has the following weakness: 1) It does not have any interactivity between clusters and outliers; 2) It favors spherical clusters.
We use $d_s(X_1, X_2)$ to represent the distance between two data points $X_1$ and $X_2$ under a certain distance metric. In a high dimensional space the data are usually sparse, and widely used distance metric such as Euclidean distance may not work well as dimensionality goes higher. The $L_p$ norm is widely used in the research work of distance measurement. $L_p$: $d(X_1, X_2) = (\sum_{i=1}^{d} |X_{1i} - X_{2i}|^p)^{1/p}$. In our previous work, we prefer $L_{0.1}$ to $L_2$ metric. For two data points $X_1$ and $X_2$, under $L_{0.1}$ norm, their distance under a $d$-dimensional data space could be: $L_{0.1}$: $d(X_1, X_2) = (\sum_{i=1}^{d} |X_{1i} - X_{2i}|^{0.1})^{10}$. However, since in our approach we focus on working on clustering and outlier detection in individual subsets of subspaces, it’s crucial that the distance measurements in different subspaces are fair to each other. Hence we modified the $L_{0.1}$ norm slightly as $L_{0.1}'$. For two data points $X_1$ and $X_2$, under $L_{0.1}'$ norm, their distance under a $d$-dimensional data space could be: $d(X_1, X_2) = (\sum_{i=1}^{d} \frac{|X_{1i} - X_{2i}|^{0.1}}{d})^{10}$.

Suppose the distance is calculated in subspace $s$, we denote it as: $d_s(X_1, X_2)$.

The definitions are modified according to the subspace specification:

**Definition 1:** For a cluster $C_i$, let $s_{c_i}$ be its associated subspace, let MST($C$) be a minimum spanning tree on the dense cells of the minimal subgraph containing $C_i$. The *internal connecting distance (ICD)* of $C_i$, denoted as $ICD(C_i)$, is defined as the length of a longest edge of MST($C_i$). The *external connecting distance (ECD)* of $C_i$, denoted as $ECD(C_i)$, is defined as the length of a shortest edge connecting the centers of $C_i$ and other clusters.
**compactness** of $C_i$, denoted as $\text{Compactness}(C_i)$, is defined as

\begin{equation}
\text{Compactness}(C_i) = \frac{ECD(C_i)}{ICD(C_i)}.
\end{equation}

In the following we use $CPT(C_i)$ to denote $\text{Compactness}(C_i)$.

**Definition 2:** The diversity between a cluster $C$ and an outlier $O$ is defined as:

\begin{equation}
D_1(C, O) = \frac{w_1 \cdot d_{\text{min}}(O, C) + w_2 \cdot d_{\text{avr}}(O, C)}{1 + |\cos \theta|}
\end{equation}

where $w_1 = \frac{1}{CPT(C)+1}$, $w_2 = \frac{CPT(C)}{CPT(C)+1}$, $d_{\text{avr}}(O, C) = d_s(O, m_c)$, $d_{\text{min}}(O, C) = \max(d_s(O, m_c) - r_{\text{max}}, 0)$ where $s = s_c \cap s_o$, $r_{\text{max}}$ is the maximum distance of the data points in $C$ from its centroid, and $\theta$ is the angle between the eigenvector of cluster $C$ corresponding to the largest eigenvalue and the vector connecting $m_c$ and $O$. The criteria for setting the weights $w_1$ and $w_2$ are similar to those in [25].

**Definition 3:** The diversity between two clusters $C_1$ and $C_2$ is defined as:

\begin{equation}
D_2(C_1, C_2) = \frac{d_s(C_1, C_2) \ast (1 + |\cos \theta|)}{CPT(C_1) + CPT(C_2)}
\end{equation}

where $d_s(C_1, C_2)$ can be either the average distance between the two clusters or the minimum distance between them in subspace $s = s_{c_1} \cap s_{c_2}$. Here we simply apply the former one $d(m_{C_1}, m_{C_2})$. $\theta$ is the angle between the two eigenvectors corresponding to the two largest eigenvalues of $C_1$ and $C_2$, respectively. The larger the value of $D_2(C_1, C_2)$ is, the larger diversity the clusters $C_1$ and $C_2$ have to each other.
Definition 4: The diversity between two outliers $O_1$ and $O_2$ is defined as:

\[ D_3(O_1, O_2) = d_s(O_1, O_2) \]

where $s = s_{O_1} \cap s_{O_2}$

Definition 5: We measure the quality of a cluster $C$ as:

\[ Q_c(C) = \frac{\sum_{C' \in C, C' \neq C} D_2(C, C')}{k_{c} - 1} + \frac{\sum_{O \in C} D_1(C, O)}{k_o} \]

The larger $Q_c(C)$ is, the better quality cluster $C$ has.

Similarly, the quality of an outlier $O$ is reflected not only by the diversity between it and clusters, but also by the diversity between it and other outliers. The farther distances it has from other outliers and clusters, the better quality it should obtain.

Definition 6: We measure the quality of an outlier $O$ as:

\[ Q_o(O) = \frac{\sum_{O' \in O, O' \neq O} D_3(O, O')}{k_o - 1} + \frac{\sum_{C \in E} D_1(C, O)}{k_c} \]

The larger $Q_o(O)$ is, the better quality outlier $O$ has.

The main goal of the subCOID algorithm is to mine the optimal set of clusters and outliers for the input data set in cluster/outlier associated subspaces. Once we determine the set of
initial clusters and the set of initial outliers, we shall also calculate the original subspace associated with each cluster/outlier.

For a cluster $C_i \in \mathcal{C}$, we adopt the concept in PROCLUS. We observe the distribution of the data points belong to $C_i$ on each dimension, and select those which have smaller deviation as the initial subspace (subset of dimensions) of $C_i$. For an outlier $O_j \in \emptyset$, we find the cluster $C_l \in \mathcal{C}$ which is farthest to $O_j$, and set its initial subspace as the same of $C_l$.

In the mergence process, suppose in one iteration, two clusters $C_i$ and $C_j$ are merged to form a new cluster $C_{ij}$, the compactness of $C_{ij}$ is calculated, the its associated subspace is also calculated based on the subspaces associated with cluster $C_i$ and $C_j$. The subspace of the merged cluster is the union of the subsets of the dimensions of the two original clusters. The distance between two clusters $C_i$ and $C_j$ is based on the diversity measurement $D_2(C_i, C_j)$ of two clusters defined above.

### 7.2.2 dynamically insertion for indexing structure of time-related multi-dimensional data

In chapter 6, we present ClusterTree+ for time-related data. Dynamic insertion of high dimensional data becomes a critical issue in real high dimensional databases. We would like to continue the research of ClusterTree+ indexing structure, focusing on efficient and
effective approaches of dynamic insertion which can keep the structure always in optimal status.

A new coming data point in ClusterTree+ can be classified into one of three categories:

- **Cluster points**: are either the duplicates of or very close to some data points in a cluster within a given threshold.

- **Close-by points**: are the data points which are neighbors to some points in the clusters within a given threshold.

- **Random points**: are the data points which are either far away from all of the clusters and can not be bounded by any bounding sphere of the ClusterTree, or might be included in the bounding spheres of the clusters at each level, but they do not have any neighboring cluster points within a given threshold.

Different insertion strategies are designed for each type of new data points. The neighbor information of the new data point is available by applying the query process of the new data point on the ClusterTree+ so that the new data point can be classified into one of the three categories mentioned above. We can directly insert the cluster points into a certain leaf node of the structure without any modification of the tree structure. As for the close-by points, they will be inserted into a certain leaf node of the structure which contains its nearest neighbors, but the radius and centroid of the certain node should be slightly
adjusted. If a new coming data point is a random point, there are two cases: if absolutely no nodes contain it, it will be collected for creating another ClusterTree+ later; if it can be bounded by bounding spheres of some clusters in the ClusterTree+, but they don’t have any neighboring cluster points in it, it should be treated differently.

For the latter case of the random points, a term \textit{maximum inclusion depth} is introduced in ClusterTree which means the lowest level of the subclusters whose bounding sphere contains the random point. ClusterTree proposes a \textit{delayed insertion} approach which stores a single data point without changing the radius and centroid of any node in the ClusterTree, and optimizes the new inserted data points when the amount of random points reaches a certain threshold.

However, there are two major problems in the random points processing mentioned above:

- the threshold of the amount of random points to optimize the current cluster is hard to decide for the complicated cases of the clusters. In a cluster, if the new coming random points are all scattered sparsely in the bounding sphere of the cluster, the original subclusters still can efficiently and effectively represent the correct status of this cluster even though the amount of the new coming random points is very large. On the other hand, if the new coming random points are close to each other, the status of the current cluster should be updated even though the total amount of the new random points is not very large. So the threshold could not be simply some
number such as the percent of the whole data points of the current cluster.

- the cluster reorganization process is quite time-consuming when more nodes and data points are involved. And the reorganization could be recursively propagated upward to the top node of the whole tree structure. So we should avoid the reorganization cases as much as possible.

According to these concerns, we focus on the dynamic insertion of the second case of the random points. The random points mentioned below are of this kind.

There are two ways to implement the dynamic insertion issue in our structure:

- based on the partition of the bounding sphere of the current cluster. It’s possible that we separate the “space” of the cluster into several parts which have different possibility of welcoming new coming random points. Of course, the “welcome possibility” of those areas occupied by some subclusters is 0 since those new data points in such areas are either cluster points or close-by points. Based on the possibility level of different areas and some presumptions, maybe we can predict the positions of the new coming data points and further determine if current inserted new data point is an outlier or “genuine” one. The problem of this approach is that the bounding spaces of those clusters containing high dimensional data points are normally very sparse, so the efficiency and effectiveness can not be guaranteed.
based on the new coming random points themselves to figure out some techniques
to improve the performance of dynamic insertion. This is a more reasonable and
practicable one. Our design is based on this point of view.

Definition (pseudo-cluster): a group of the random points in a certain cluster which are
close to each other. This term is helpful for the further analysis and processing of the
dynamic insertion.

We modify the structure of ClusterTree*. It has two kinds of nodes: internal and leaf nodes.
The internal nodes are defined as:

\[(7.2.7)\]

\[
\text{Node} : [\text{Node}\_id, \gamma, (\text{Entry}_1, \text{Entry}_2, \cdots , \text{Entry}_\gamma), \text{RPS}, \text{CM}] \ (m_{\text{node}} \leq \gamma \leq M_{\text{node}}),
\]

\[
\text{Entry}_i : (\text{SC}_i, \text{BS}_i, \text{SN}_i),
\]

where \(\text{Node}\_id\) is the node identifier, \(\gamma\) is the number of the entries in the node, \(m_{\text{node}}\) and
\(M_{\text{node}}\) define the minimum and maximum numbers of entries in the node. An entry is
created for each subcluster of the cluster for which the current non-leaf node represents.

In entry \(\text{Entry}_i\), \(\text{SC}_i\) is a pointer to the \(i\)-th subcluster, \(\text{BS}_i\) is the bounding sphere for the
subcluster and \(\text{SN}_i\) is the number of data points in the \(i\)-th subcluster.

The dynamic insertion part in the internal nodes includes RPS and CM. The RPS is the
structure containing the random points information:

\[
RPS : [RP_1, RP_2, \cdots],
\]

\[
RP_i : (ADR_i, PC_i),
\]  

(7.2.8)

where ADR\(_i\) is the address of the random point residing on the secondary storage, and PC\(_i\) is the pseudo-cluster number it is in. We will explain it later.

The CM is the correlation matrix of subclusters and pseudo-clusters of the current cluster. It stores the amount of common neighbor data points of each (subcluster, pseudo-cluster) pair and each (pseudo-cluster, pseudo-cluster) pair. Also it stores the amount of random points each pseudo-cluster contains and the amount of “original” data points each subcluster contains.

The basic idea is that we should avoid reorganization as much as possible since it is very time-consuming. When a new random point comes to the suitable subcluster with its maximum inclusion depth, it is firstly inserted into this subcluster node’s RPS structure with the pseudo-cluster number of 0. If it is close enough to a previously random points, a new pseudo-cluster is created including both of them; if it is close enough to several random points, it can be a common neighbor point of more than one pseudo-clusters. We inspect the correlations of the (subcluster, pseudo-cluster) pairs and those of the (pseudo-cluster, pseudo-cluster) pairs. If the amount of data points which are the common neighbors of two sides exceeds certain threshold, we regard these two structures as being ready for mergence.
7.2.3 combination of fuzzy clustering and shrinking-based data analysis approaches.

Some data in the real world are not naturally well organized. Clusters in the data may overlap each other. Fuzzy concept can be applied to further smooth the shrinking movement of the "boundary" data points.

The concept of fuzzy sets was first introduced by Zadeh (1965) to represent vagueness. The use of fuzzy set theory is becoming popular because it produces not only crisp decision when necessary but also corresponding degree of membership. Usually, membership functions are defined based on a distance function, such that membership degrees express proximities of entities to cluster centers. In conventional clustering, sample is either assigned to or not assigned to a group. Assigning each data point to exactly one cluster often causes problems, because in real world problems a crisp separation of clusters is rarely possible due to overlapping of classes. Also there are exceptions which cannot be suitably assigned to any cluster. Fuzzy sets extend to clustering in that object of the data set may be fractionally assigned to multiple clusters, that is, each point of data set belongs to groups by a membership function. This allows for ambiguity in the data and yields detailed information about the structure of the data, and the algorithms adapt to noisy data and classes that are not well separated. Most fuzzy cluster analysis methods optimize a subjective function that evaluates a given fuzzy assignment of data to clusters.
One of the classic fuzzy clustering approach is the Fuzzy C-means Method designed by Dunn, J.C., Bezdek, J. C. In brief, for a data set $X$ with size of $n$ and cluster number of $c$, it extends the classical within groups sum of squared error objective function to a fuzzy version by minimizing the objective function with weighting exponent $m$, $1 \leq m < \infty$:

$$J_m(U, V) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m d^2(x_k, v_i),$$

where $U$ is a partition of $X$ in $c$ part, $V = v = (v_1, v_2, ..., v_c)$ are the cluster centers in $\mathbb{R}^p$, and $A$ is any $(p \times p)$ symmetric positive definite matrix defined as the following:

$$d(x_k, v_i) = \sqrt{(x_k - v_i)^\top (x_k - v_i)},$$

where $d(x_k, v_i)$ is an inner product induced norm on $\mathbb{R}^p$, $u_{ik}$ is referred to as the grade of membership of $x_k$ to the cluster $i$. This grade of membership satisfies the following constraints:

$$0 \leq u_{ik} \leq 1, \quad \text{for } 1 \leq i \leq c, 1 \leq k \leq n,$$

$$0 < \sum_{k=1}^{n} u_{ik} < n, \quad \text{for } 1 \leq i \leq c,$$

$$\sum_{i=1}^{c} u_{ik} = 1, \quad \text{for } 1 \leq k \leq n.$$
The fuzzy C-Means (FCM) uses an iterative optimization of the objective function, based on the weighted similarity measure between \( x_k \) and the cluster center \( v_i \). During each iteration, it calculates the \( c \) cluster centers \( \{v_{i,t}\}, i = 1, \ldots, c \)

\[
v_{i,t} = \frac{\sum_{k=1}^{n} u_{ik,t-1}^m x_k}{\sum_{k=1}^{n} u_{ik,t-1}^m},
\]

for those data points not of any current cluster center, it calculate the following

\[
u_{ik,t} = \frac{1}{\sum_{j=1}^{c} \left( \frac{d_{ik,t}}{d_{ij,t}} \right)^{m-1}},
\]

When a predefined termination condition is satisfied, the algorithm is terminated.

One of the advantages of the shrinking-based clustering approach has over other clustering approaches is the subtlety characteristics of data point movement. In each iteration step, the data points move gradually according to the gravitation of neighboring data points. Data points won’t move dramatically in one step which may easily cause improper movement such as unsuitable direction or unfavorable amplitude.

One of the critical aspect of the data point movement is the way to handle “boundary data points”. Here “boundary” means that those data points are among the “intersection area” of multiple clusters.

It can be easily understood that the movement accuracy of boundary data points is more
important than that of center data points. For center data points, even though they move a little bit improperly during one iteration, they will still be in the center area of a cluster, whereas for boundary data points, incorrect movement during one iteration might lead them to the wrong cluster. Thus more effort should be put on the improvement of the movement of boundary data points. We tend to use fuzzy clustering concept to further smoothen the boundary data point movement in shrinking-based approaches.
Bibliography


