A Study of Efficient and Robust Clustering for Large-Scale Datasets

Hao Liu

A Dissertation

Submitted to the Division of Synergetic Information Science

in partial fulfillment of the requirements for the

Doctor of Philosophy

at the

Graduate School of Information Science and Technology

HOKKAIDO UNIVERSITY

June 2014

A Study of Efficient and Robust Clustering for Large-Scale Datasets

Hao Liu

Abstract

Clustering, also called cluster analysis, is one of the most important techniques for analyzing data, and has been widely applied in many fields, such as data mining, machine learning, pattern recognition, information retrieval, etc. However, the amount of data is exponentially increasing because of the recent big data explosion. Many excellent clustering algorithms designed with good robustness, e.g. the recently proposed fuzzy density-based clustering algorithms, are not efficient enough for large-scale datasets. On the other hand, some efficient algorithms, e.g. the incremental clustering approaches, are not robust enough when they learn new data, especially in a noisy environment. In this dissertation, two novel methods for efficient and robust clustering are proposed. One is an efficient fuzzy density-based clustering algorithm which uses a new concept, called "landmark", to obtain a good representation for the input dataset. Since the number of landmarks is much smaller than that of the actual data, the computational cost is significantly reduced. The other one is a robust incremental self-organizing neural network, where a new flexible network structure is modeled so that the algorithm is able to automatically determine a suitable number of neurons for the network while keeping stability to outliers.

In Chapter 1, the background of this study, including the purpose of clustering and a brief summarization of the state of the art, is introduced. Then the challenge of detecting clusters in large-scale datasets is discussed. After that, the purpose of this study and a brief introduction of the two proposed methods as well as a description of their contributions are presented.

Chapter 2 presents the fundamentals necessary for the remaining chapters. Three classical clustering techniques, i.e. partitional, hierarchical and density-based clusterings, are described and their benefits and drawbacks are discussed. Then it is followed by the discussion on incremental clustering, including classical ones and relatively new ones based on the self-organizing neural networks.

In Chapter 3, as the base of this study, a density-based clustering method, called Landmark Fuzzy Neighborhood DBSCAN (landmark FN-DBSCAN), is presented, aiming to improve the efficiency of FN-DBSCAN. The new concept, landmark, is introduced to represent a subset of the input dataset so that the original dataset can be compressed and represented by the generated landmarks. We give a theoretical analysis on the time and space complexities of the algorithm, which shows that both of them are linear to the size of the dataset. The experiments presented in this chapter also show that the landmark FN-DBSCAN algorithm can be much faster than FN-DBSCAN, while maintaining the good clustering quality of the original algorithm. However, we will see that this algorithm has a critical limitation, because the final

clustering result may be easily influenced by the input data orders, if the number of landmarks is very small.

Chapter 4 introduces a new effective and efficient solution for the task of landmark generation, i.e. Locally Adaptive Incremental LBG (LAI-LBG). In order to overcome the above-mentioned drawback, LAI-LBG incrementally inserts and removes landmarks with no requirement of predefining the number of landmarks, and it is not sensitive to the initializations and the input data orders, which means that the algorithm can pursue a well-balanced representation of the input dataset with a small number of landmarks. In addition, it has a higher efficiency than its competitors with comparable quantization quality.

In Chapter 5, a novel density-based clustering algorithm which is a significant improvement of the landmark FN-DBSCAN algorithm is proposed. Benefiting from LAI-LBG, the new algorithm can obtain a good clustering result with a few number of landmarks, which indicates that the efficiency has further increased. In addition, it is not sensitive to the input data orders, which means that the robustness is improved. Furthermore, its parameters have typical values or can be automatically estimated, which means that the practicability is also enhanced.

Chapter 6 presents a novel incremental self-organizing neural network, called enHanced Incremental Growing Neural Gas (Hi-GNG). In this work, an enhancement of the conventional competitive Hebbian learning (enhanced CHL) is also proposed, where the birth and death of neurons, the creation and elimination of connections between each pair of neurons, and the adjustment of the connection-strength of each connection are modeled. On the basis of the enhanced CHL, the Hi-GNG algorithm can automatically generate a topological structure with a suitable number of neurons. Furthermore, it is also robust to noisy data and even more efficient than its competitors, such as the Growing Neural Gas (GNG) algorithm.

In Chapter 7, the methods proposed in this dissertation are summarized and some future works are discussed.

Acknowledgments

I would like to express my sincerest gratitude to my respected academic supervisor, Prof. Masahito Kurihara, for all his kind help, inspiring suggestions, patient guidance, and encouraging words that I will never forget. I feel so lucky that I could be one of his students for the past three years. I sincerely wish to thank Prof. Satoshi Oyama for all his kind directions for my research work and the opportunities that he gave to me to review papers for excellent conferences, which makes me more mature as a researcher in the academic activities. I am also grateful to Prof. Haruhiko Sato, a friend-like assistant professor of our laboratory, for all his kind support. The experience of attending conferences, held in Hong Kong and Seoul, Korea, with him are my happy memories. In addition, I would like to thank Prof. Keiji Suzuki, Prof. Tetsuo Ono and Prof. Masahito Yamamoto for their helpful comments and suggestions on the improvements of this dissertation.

I would like to give my special thanks to my good friend Mr. Zhuoran Xu, a Ph.D. candidate of Hokkaido University, for his discussions on clustering and neural networks, helps in the experiments, support in my daily life and a friendship over 10-years. I also feel lucky that I can meet with Ms. Huang at Hokkaido University and the days that I shared with her are my precious memories. I intend to thank Mr. Banno, Ms. Horimiya, Mr. Ding for all their warm help and support during the period when I first came to Japan. Thank you all my friends: Mr. and Mrs. He, Mr. and Mrs. Zhang, Mr. Duan, Ms. Hu, Ms. Song, Mr. Ji, Mr. Chen, Mr. Katanosaka, Mr. Bando, Mr. Yamashita, Mr. Narita and all the members in my laboratory, without you, my life in Japan cannot be so happy and colorful.

I also feel deeply grateful to Prof. Xiaojuan Ban, who led me to the world of being a researcher when I studied in University of Science and Technology, Beijing. In addition, I would like to thank my best friend, Dr. Shi, for all his support and encouragement for the past 10 years.

I would like to take this opportunity to express my sincere gratitude to China Scholarship Council (CSC) and Hokkaido University for their financial support, which

makes possible for me to study aboard. I would also like to thank IEEE computational society and the International Federation of Automatic Control (IFAC) for their kind financial support to my participation in IJCNN 2013 and IFAC World Congress 2011, respectively.

Finally, to my dear parents. I always owe a big "thank you" to you for your unconditional support. I love you!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In recent times, people are living in a world full of data. The rapid development of information science and technology allows us to collect, store or analyze data in an effective and convenient way that we could never imagine before. The impact on our lives is inescapable [1, 2]. Clustering, also called cluster analysis, is one of the most important techniques for analyzing data. It assumes that a set of observations or patterns which have similar characteristics are forming natural grouping(s) in the feature space, which can be observed by some statistical classification techniques. Webster (Merriam-Webster Online Dictionary) [3] gives a definition of clustering as "a statistical classification technique for discovering whether the individuals of a population fall into different groups by making quantitative comparisons of multiple characteristics".

On the basis of a specific similarity measure, a clustering algorithm is a computer program which can automatically divide a given dataset into several groups, named clusters, where the data points in the same cluster are more similar to each other than to those in different clusters. From a data modeling perspective, the long history of clustering roots from mathematics, statistics and numerical analysis. Standing on the point of view of machine learning, clustering is an unsupervised learning technique which requires the algorithm to have a capability of processing datasets without any prior knowledge. Since its fundamental and irreplaceable contributions to data analysis, clustering now has been widely applied in many scientific fields including data mining, machine learning, pattern recognition, information retrieval, etc.

During the past decades, many excellent clustering algorithms have been proposed, which have been successfully applied as solutions for different kinds of clustering or classification problems. A rough but widely agreed framework is to classify the existing clustering techniques into partitional clustering, hierarchical clustering or density-based clustering [4].

Partitional clustering classifies a dataset into a set of disjoint non-empty clusters. A classical technique in this category is the k-means clustering [5]. It assumes that a cluster distributes with a single point as the center. Given n data points and k initialized clusters, the standard k-means clustering algorithm calculates the center for each cluster and reassign all data points to the cluster with the closest center. In general, this process is repeated until the Mean Squared Error (MSE) converges. Since k-means is simple and efficient, it is still widely used even though over 50 years have passed since it was first proposed [6]. Motivated by k-means, many improved algorithms have been developed [7, 8, 9, 10, 11]. However, there are several shortcomings of partitional clustering techniques, e.g. the number of clusters is a prerequisite; the clustering results may significantly depend on the initialization; and complex-shaped clusters are often difficult to detect.

Hierarchical clustering, in general, classifies data points with a hierarchical structure which is usually implemented by a binary clustering tree or a dendrogram [12]. Hierarchical clustering can be further categorized by the ways of how to organize data points: by agglomeration or division [13]. A standard agglomerative clustering algorithm is initialized with N clusters, where N usually equals to the number of data points in the dataset. By merging the closest clusters into one single cluster, the number of clusters decreases until some stop condition is reached. On the other hand, a classical divisive hierarchical clustering algorithm usually starts with one cluster containing all data points, splitting the cluster repeatedly to make a collection of smaller clusters until some predefined stop condition is met. An important step of hierarchical clustering is to choose a similarity measure which can be, but not limited to, single-linkage [14], complete-linkage [15] or average-linkage [16]. Using different similarity measures may lead to different clustering results. The main difficulty of

using hierarchical clustering is their low efficiency for large-scale datasets. Given n data points, a typical agglomerative clustering requires $O(n^3)$ time complexity and a standard divisive clustering usually costs $O(2^n)$. Some approaches were proposed to improve the efficiency, e.g. the literatures [17, 18], but they still need at least $O(n^2)$ time complexity [4].

Some hybrid algorithms which combine the advantages of partitional and hierarchical clusterings have been proposed, such as, Cohesion-based Self-Merging (CSM) [10] and CHAMELEON [19]. CSM is a two-phase clustering algorithm which partitions a dataset into several small sub-clusters in the first phase, and then merges these sub-clusters in the second phase based on a similarity measure of the inter-cluster distances in a hierarchical manner. CHAMELEON is another well-known hybrid clustering algorithm. Different from the typical hierarchical clustering algorithms, CHAMELEON clusters a dataset by considering both interconnectivity (the number of links between two clusters) and closeness (the length of those links) in identifying the most similar pair of clusters, which makes CHAMELEON powerful in discovering clusters in arbitrary shapes and of different sizes.

Density-based clustering algorithms are proposed on the basis of several density-related concepts, e.g. *ε-neighborhood*, *core-point*, *directly density-reachable*, *density-reachable* and *density-connected* [20]. DBSCAN [21] is a typical one implemented with these features. It assumes that the probability density of a small area in the feature space is uniform and the data density in each desired cluster is higher than the outside of the cluster. In general, compared to the other categories of clustering techniques, density-based clustering techniques usually have several advantages, such as no requirement for the number of clusters, the ability of detecting arbitrary shaped clusters and the effectiveness of outliers detection. However, they often suffer a difficulty in the parameter selection. The main reason is that a typical density-based clustering algorithm, e.g. DBSCAN, detects local density values by counting the number of data in a so-called crisp-neighborhood which is directly controlled by the parameter ε (or *Eps*). In addition, to obtain the *core-points*, the counted number must exceed a threshold which is defined by another parameter, *MinPTS*. Unfortu-

nately, in practice, it is very hard to select appropriate values for them. Recently, building a soft model for the neighborhood has attracted a lot of attention. Fuzzy Joint Point (FJP) and its extension, Noise Robust FJP (NRFJP), introduced the concept of fuzzy-neighborhood into density-based clustering algorithms and partially solved this problem [22, 23]. Unfortunately, a FJP-like method usually requires a very high time complexity. Another approach, Fuzzy-Neighborhood DBSCAN (FN-DBSCAN) provides a good balance between clustering efficiency and quality [24]. The key idea of FN-DBSCAN is to use fuzzy neighborhood functions to define a new fuzzy neighborhood instead of the old crisp neighborhood used in DBSCAN. However, FN-DBSCAN has a typical time complexity of $O(n^2)$, which means that it is not suitable to deal with large-scale datasets. Like the classical DBSCAN algorithm, FN-DBSCAN can also be extended with a spatial index support, e.g. R^*-tree [25] or *kd*-tree [26], so that the average time complexity can be reduced to $O(n * log(n))$ [21].

However, due to the recent big data explosion, an algorithm with a time complexity of $O(n^2)$ or even $O(n * log(n))$ may not be efficient enough if the input data scale is very large. For example, in a geographic information system (GIS), a raster image of an earth area of 1,000 by 1,000 meters captured by the NOAA satellite with Advanced Very High Resolution Radiometer (AVHRR) sensor may generate over four millions of data from each spectral channel [27]. In the social networks over the internet, in every minute, Facebook users update over 246,000 statuses and about 278,000 tweets appear in the Twiter website [28]. People are seeking to find useful information from such a huge amount of data. Therefore, designing more efficient and robust clustering algorithms is still worth more deep research to catch up with this rapidly developing world. Nowadays, there are mainly two approaches to improve the clustering efficiency. One is to improve the algorithm so that the computational cost can be directly reduced. The other one is to use an incremental manner for clustering so that the algorithm can classify new data into clusters without recalculation of the previously processed data. Regarding these two approaches, in this dissertation, we propose two novel approaches to an efficient and robust density-based clustering and an incremental self-organizing neural network, respectively.

Here, we summarize the main features of the proposed methods and explain why we need these features as follows.

1. The algorithms have a significantly higher efficiency than the other comparable algorithms. According to the previously discussed big data explosion, the efficiency of a clustering algorithm is highly more important than ever before. This new challenge is encouraging researchers to find new efficient ways to analyze data [29].

2. The algorithms do not require the number of clusters as a prerequisite. One of the biggest problems with many classical clustering algorithms, e.g. most partitional clustering algorithms, is that they require the number of clusters, k, as an input parameter [6]. In this case, users have to own expertise and select an accurate number for the value of k. Although there are some estimation methods for determining k [30, 31, 32], it is very common that the value of k cannot be exactly predicted, which often directly causes a low quality of data analysis.

3. The algorithms can discover the clusters with arbitrary shapes. With many clustering algorithms, the shapes of the detected clusters are limited to convex shapes, e.g. spheres, polygons, etc. But a cluster, especially in a spatial dataset, may not have a convex shape, but may be represented in an arbitrary shape [33]. Moreover, the shape of a cluster might be more complicated in a higher-dimensional space.

4. The algorithms is insensitive to outliers and can eliminate them if necessary. It is very common that a given dataset contains some outliers [21]. In general, it is preferable that a clustering algorithm is insensitive to the outliers and has a capability of removing such exceptional data.

5. The algorithms have typical values for the parameters or it can automatically estimate appropriate values. According to the previous discussions, density-based algorithms are usually suffering a difficulty in selecting appropriate parameter

values. Since the proposed algorithm is also a density-based approach, it will significantly improve the practicability of a clustering algorithm, if the parameters can be automatically determined or set by typical values.

The rest of this dissertation is organized as follows. We summarize the state of the art of clustering techniques in Chapter 2. Next, we present our preliminary work which is a density-based clustering algorithm called landmark fuzzy neighborhood DBSCAN (landmark FN-DBSCAN) in Chapter 3. After that, we present a new landmark generation method called Locally Adaptive Incremental LBG (LAI-LBG) in Chapter 4. On the basis of previous two algorithms, we propose a novel density-based clustering in Chapter 5 as one of main contributions in this dissertation. In Chapter 6, we propose another main contribution which is a new approach for incremental clustering, called enhanced incremental growing neural gas (Hi-GNG). Finally, we give the conclusions and discuss about our future work in Chapter 7.

Chapter 2

Fundamentals

Designing a robust and efficient clustering algorithm has been attracting a lot of attention for the past decades. In general, there are two ways of categorizing the existing clustering techniques. One is a widely agreed framework according to the main characteristic in the procedure of clustering, i.e. partitional clustering, hierarchical clustering and density-based clustering [4]. The other one is to categorize clustering techniques by the manner of processing data, i.e. batch clustering and incremental clustering.

In this chapter, we summarize the state of the art of the clustering techniques for the categories of partitional clustering, hierarchical clustering and density-based clustering. In addition, in each mentioned clustering category, we consider both batch and incremental clustering approaches.

2.1 Classical Clustering Techniques

2.1.1 Partitional Clustering

The partitional clustering techniques can be further classified into hard partitional clustering and fuzzy partitional clustering.

Hard Partitional Clustering

In hard partitional clustering, each data belongs to and must only belong to one cluster. Consider a m-dimensional dataset $X = \{x_i\}, i = 1, 2, ..., n$, where $x_i = (x_{i1}, x_{i2}, ..., x_{im}) \in \Re^m$ and a set of clusters $C = \{C_j\}, j = 1, ..., k$. The basic idea of hard partitional clustering is to classify X into a set of disjoint non-empty clusters [4], i.e.:

1. $C_i \cap C_j = \emptyset$, where $C_i, C_j \neq \emptyset$, $i, j = 1, ..., k, i \neq j$;
2. $\bigcup_{i=1}^{k} C_i = X$;

The classical k-means algorithm is one of the most successful approach in this category. It assumes that a cluster is distributed with a single point as the center. Let V_k be the mean of the cluster C_k. A squared error function (Equation (2.1)) is used to measure the fitness between V_k and the data in the cluster, C_k.

$$E(C_k) = \sum_{x_i \in c_k} d(x_i, V_k)^2 \tag{2.1}$$

where, in general, the function $d(x', x)$ is the Euclidean distance between two m-dimensional vectors $x' = (x'_1, x'_2, ..., x'_m)$ and $x = (x_1, x_2, ..., x_m)$, i.e., $d(x', x) = \sqrt{\sum_{i=1}^{m} (x'_i - x_i)^2}$.

Since there are k clusters with k centers, the objective function which is usually the Mean Squared Error (MSE) (Equation (2.2)), is used to measure the distortions of the optimization by the algorithm.

$$D = \frac{1}{n} \sum_{j=1}^{k} E(C_j) = \frac{1}{n} \sum_{j=1}^{k} \sum_{x_i \in c_k} d(x_i, V_j)^2 \tag{2.2}$$

Given a dataset and k initialized clusters, the standard k-means clustering algorithm classifies data into k clusters by minimizing MSE as Algorithm 1 shows:

However, some limitations are known in the standard k-means algorithm.

First, the sensitivity to the initializations. The k-means clustering starts with k arbitrary centers. Unfortunately, the final clustering quality significantly depends

Algorithm 1 The k-means Clustering Algorithm

Input: X, k

Output: k partitions of X

1: Randomly initialize k vectors as the centers of k clusters;
2: Assign each data to its closest cluster by calculating the distance between the data and the cluster centers;
3: Repeat Step 2 until the assignments of all data do not change, i.e. MSE is minimized.

on the initialization. In general, a bad initialization may results in a seriously wrong clustering result. David A. and Sergei V. proposed an initialization algorithm called k-means++ [34], which guarantees a good solution with a time complexity of $O(log(k))$.

Second, the problem of local minimum. The optimization by minimizing MSE is known as an NP-hard problem (even for k=2). From Step 2 of Algorithm 1, we know that k-means takes a greedy strategy to assign data to clusters. Since the best partition of a dataset usually falls in an exponentially large solution space, sometimes, such a greedy way easily results in a convergence to a local minimum. Furthermore, a bad initialization may lead to a deterioration of this situation [6].

Third, the sensitivity to the outliers. Outliers usually exist quite far away from normal data, which means that they have a significantly higher or lower value than the normal data. In this case, one single outlier may heavily influence the result of the calculations for the cluster centers. The k-medoids [35], a variation of k-means, chooses the medoid instead of calculating the center for each cluster, where a medoid is usually a data in the dataset. By minimizing the sum of pairwise dissimilarities instead of MSE, the k-medoids clustering algorithm can perform more robust to the outliers than the original k-means clustering algorithm.

Due to the simplicity of implementation and efficiency in practice, the k-means clustering is still widely used even though over 50 years have passed since it was first proposed [6]. Inspired by k-means, many variations of k-means have been developed. The Iterative Self-Organizing Data Analysis Technique (ISODATA) [36] merges clusters, the separation distance of which in multi-spectral feature space is less than a user-specified parameter, and also split a big cluster into two smaller clusters if the standard deviation is greater than a user-specified parameter. Kanungo T. et al.

proposed a k-means implementation with an efficient manner [37]. Their algorithm organizes all the data in a kd-tree structure so that all the data which are closest to a specified given data can be efficiently found out. The Clustering Large Applications based on RANdomized Search (CLARANS) is an improvement of the k-medoid algorithm, which is more effective and efficient. More hard partitional clustering algorithms can be found in literatures [7, 8, 9, 10, 11, 38, 39].

Fuzzy Partitional Clustering

Unlike hard partitional clustering, fuzzy partitional clustering allows each data to be assigned to more than one clusters, and each cluster is associated with all data points, which is measured by a set of membership degrees. A representative approach is fuzzy c-means (FCM) (first developed by Dunn [40] and later improved by Bezdek J.C. [41, 42]). The weighting factors of FCM are adjustable and the distance measure is also exchangeable so that the sensitivity to noisy data can be essentially controlled. The FCM algorithm uses Equation (2.3) instead of MSE.

$$J_m = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^r \|x_i - V_j\|^2, 1 \leq r \leq \infty \tag{2.3}$$

where k is the number of clusters, r is any real number greater than 1, u_{ij} is the membership degree of x_i belonging to the cluster C_j, V_j is the center of C_j, and $\| * \|$ can be any similarity measure, e.g. Minkowski distance or Euclidean distance. In each iteration, minimize this objective function by updating the cluster centers and the membership degrees. Given a data x_i, FCM updates the membership degree of x_i belonging to cluster C_j by Equation (2.4), and updates the centers by Equation (2.5), respectively.

$$u_{ij} = \frac{1}{\sum_{p=1}^{k} (\frac{(\|x_i - V_j\|}{(\|x_i - V_p\|})^{2/(r-1)}} \tag{2.4}$$

$$V_j = \frac{\sum_{i=1}^{n} u_{ij}^r * x_i}{\sum_{i=1}^{n} u_{ij}^r} \tag{2.5}$$

Here we summarize the FCM algorithm in Algorithm 2.

Algorithm 2 The Fuzzy c-means (FCM) Algorithm

Input: X, r, k, ε

Output: the membership degrees of all data

1: Initialize a $n \times k$ matrix to represent the membership degree $U(0) \leftarrow [u_{ij}]_{n\times k}$;

2: In each iteration, e.g. the t-th iteration, update cluster centers and membership degree matrix as follows
$V_j \leftarrow \frac{\sum_{i=1}^{n} u_{ij}(t)^r * x_i}{\sum_{i=1}^{n} u_{ij}(t)^r}$,
$u_{ij}(t+1) \leftarrow \frac{1}{\sum_{p=1}^{k} (\frac{(\|x_i - V_j\|}{(\|x_i - V_p\|})^{2/(r-1)}}$;

3: Repeat Step 2 until $\|U(t+1) - U(t)\| < \varepsilon$;

Recently, Havens, T. C., Bezdek, J.C. et al. significantly improved the efficiency of FCM so that the improved version can be well applied to the processing of very large-scale datasets[43].

2.1.2 Hierarchical Clustering

An Overview of Hierarchical Clustering

Hierarchical clustering (HC) is another widely used tool for data analysis. Compared to partitional clustering, e.g. k-means, a HC algorithm does not require the number of clusters and the initialization of k clusters. A typical HC algorithm organizes data with a hierarchical structure [4]. In practice, such a hierarchical structure can be implemented by a binary tree or a dendrogram [12] where root node represents the whole dataset and each leaf node is regarded as a single data point [4]. So, HC algorithms can be further categorized by the ways of obtaining such a hierarchical structure, i.e. bottom-up method and top-down method, which are called hierarchical agglomerative clustering (HAC) and hierarchical divisive clustering (HDC), respectively [13].

A HC algorithm initializes all data points as singleton clusters (in the HAC case) or a single cluster containing all data points (in the HDC case), then it merges or splits the most appropriate clusters until the stopping criterion is reached. Here, we summarize a standard HAC algorithm in Algorithm 3.

Early representative HC algorithms are SLINK [17] and CLINK [15] which implement single-linkage and complete-linkage (see section 2.1.2 for more detailed dis-

Algorithm 3 A Hierarchical Agglomerative Clustering Algorithm

Input: X

Output: the partitions of X

1: Initialize each data point as a singleton cluster;
2: Merge two closest clusters;
3: Repeat Step 2 until all clusters have been merged into a single cluster.

cussions), respectively. The main difficulty of using classical HC algorithms is their low efficiency for large-scale datasets. Given n data points, a typical HAC requires $O(n^3)$ time complexity and a standard HDC usually costs $O(2^n)$. A well-known HC algorithm is CURE [18] which provides a more robust solution for detecting outliers in large-scale dataset. In addition, CURE also enhanced the ability to detect non-spherical shaped clusters. ROCK uses a new concept, called *link*, to measure the similarity between a pair of data points, which makes ROCK suitable for both boolean and categorical attributes [44].

Recently, HC algorithms have been extended in many directions, one of which is a combination of partitional clustering and hierarchical clustering, which is usually called hybrid clustering. The Cohesion-based Self-Merging (CSM) algorithm [10] is a two-phase clustering algorithm which partitions a dataset into several small sub-clusters in the first phase, and then merges these sub-clusters in the second phase based on a similarity measure of the inter-cluster distances in a hierarchical manner. CHAMELEON [19] is another well-known hybrid clustering algorithm. Different from the typical hierarchical clustering algorithms, CHAMELEON clusters a dataset by considering both interconnectivity (the number of links between two clusters) and closeness (the length of those links) in identifying the most similar pair of clusters, which makes CHAMELEON powerful in discovering clusters in arbitrary shapes and of different sizes. Clustering documents by HC also attracted a lot of attention [45, 46]. Benjamin C. M. Fung et al. proposed a hierarchical document clustering where a new concept, called *frequent itemsets*, which is a set of common words in a document, is introduced to identify clusters and also used to produce a hierarchical topic tree for clusters. By focusing on *frequent items*, the dimensionality of the document set can be reduced [46]. A. Kraskov and P. Grassberger introduced a

new similarity measure, called mutual information (MI), and they also proposed the mutual information clustering (MIC) algorithm based on MI [47], which can be well applied in some real-world problems, such as constructing phylogenetic trees from mitochondrial DNA sequences and analyzing Electrocardiography (ECG) of a pregnant woman. Using parallel techniques for HC also received many attention, which are discussed in [48, 49, 50]. Another popular direction is the extension of HC algorithms with a capability of incremental clustering, which we will discuss in section 2.2.1.

More interesting recent work related to HC algorithms can be found in [14, 16, 51, 52, 53, 54].

Linkage criteria

No matter a HC algorithm is implemented with HAC or HDC, the most important thing to obtain the final clustering result is to determine a suitable linkage criterion, i.e. the similarity measure of two clusters. Using different linkage criteria may directly lead to different clustering results. Letting C_i and C_j be two clusters, we summarize several widely-used linkage criteria as follows.

Single-linkage [55] can be described by Equation (2.6).

$$D(C_i, C_j) = \min\{d(x, y), x \in C_i, y \in C_j\} \tag{2.6}$$

where we can see that single-linkage is a local-search strategy where the similarity of two clusters is measured by their most similar members. In general, single-linkage can be well applied to discover clusters with complex-shapes, but, sometimes, two clusters may be easily mis-merged if there exist some noisy data establishing a bridge between them.

Complete-linkage [56], however, is a non-local strategy to measure the similarity of two clusters given by Equation (2.7).

$$D(C_i, C_j) = \max\{d(x, y), x \in C_i, y \in C_j\} \tag{2.7}$$

Unlike single-linkage, complete-linkage measures the similarity by the most dissimilar members so that the entire cluster structure can influence merge decisions [57]. However, it often suffers from the problem with outliers.

Another well-known linkage criterion is the average-linkage criterion, also known as Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [58], which is described as follows.

$$D(C_i, C_j) = \frac{1}{|C_i| * |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y) \tag{2.8}$$

The above three linkage criteria are called graph method since they calculate all data of a cluster pair. There are also some other linkage criteria, whereas they are called geometric methods which use geometric centers to represent clusters and determine their distances. Interested readers are recommended to read the literatures [4, 59, 60, 61, 62, 48] to find more major linkage criteria.

2.1.3 Density-Based Clustering

Compared to partitional clustering and hierarchical clustering, denstiy-based clustering is a younger category of clustering techniques. The first method which is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm was proposed at ACM-KDD 1996 [21].

In general, density-based clustering algorithms are based on several density-related concepts, such as density, connectivity and boundary [60]. DBSCAN defines such concepts including *ε-neighborhood*, *core-point*, *directly density-reachable*, *density-reachable* and *density-connected* [20], so that it detects clusters by searching for the density-connected data points. The standard DBSCAN has a time complexity of $O(n^2)$ and can be further reduce to $O(n*log(n))$ with some spatial indexing structure, e.g. $R^\star$-tree or *kd*-tree. In addition, DBSCAN has only two parameters ε (or *Eps*) and *MinPts* which are used to define ε-neighborhood of a data point and a core point. Due to the effectivity and efficiency, DBSCAN has become the most widely used density-based clustering algorithm, especially for detecting clusters in spatial

datasets where data points have low-dimensionality but the amount of data is usually dramatically large. Since DBSCAN is very important to the proposed methods of this dissertation, we will detail this algorithm in Section 3.1.1.

Compared with other categories of clustering techniques, density-based clustering methods usually have the following advantages:

1. The number of clusters in a dataset is not a prerequisite of clustering.

2. The detected clusters can be represented in an arbitrary shape.

3. Outliers can be detected and removed.

However, DBSCAN often suffers from some limitations, e.g. different part of data may have different densities indicating a different determination for parameters. However, the two parameters used in DBSCAN are pre-defined and globally fixed before carrying out clustering, which results in a difficulty in the parameter selection [19]. Furthermore, the neighborhood radius is directly controlled by the parameter ε, which is a crisp-neighborhood model indicating that the algorithm is poor at representing the local densities [24]. Unfortunately, there is no straight-forward way to determine them [60]. Over the past decades, the improvements or extensions of DBSCAN has attracted a lot of attention.

The Generalized DBSCAN (GDBSCAN) is a generalized version of the DBSCAN algorithm [63]. It can use any notion of a neighborhood of a data point if the definition of the neighborhood is based on a binary predicate which is symmetric and reflexive. Second, to define the cardinality, instead of simply counting the data points in the neighborhood, GDBSCAN can use some other methods considering both spatial and the non-spatial attributes, e.g. the average income of a city. These new features make GBSCAN more powerful than the original DBSCAN algorithm to the applications in solving real world problems.

The algorithm, Ordering Points To Identify the Clustering Structure (OPTICS), extended DBSCAN with the ability to analyze clusters with different densities [64]. It can automatically or interactively build an augmented ordering of data so that

a density-based clustering structure can be represented. OPTICS uses the same parameters as DBSCAN, but it is easier to parameterize than the former one. A main lamination of OPTICS is that the clustering results are not easy to use. In general, a visualization method is needed to analyze the results. Some improvements or modifications of OPTICS can be found in literatures [65, 66, 67, 68]

In order to improve the efficiency, DENsity-based CLUstEring (DENCLUE) [69] and its improvement DENCLUE 2.0 [70] employ a cluster model based on kernel density estimation. A cluster is defined by a local maximum of the estimated density function. Data points are assigned to the same cluster if they are going to the same local maximum, using the procedure called hill climbing. The *rough*-DBSCAN algorithm is another good modification of DBSCAN regarding the same objective, which uses a modified *Leaders* algorithm to extract a small part of data points and then use DBSCAN to classify the leaders so that the efficiency can be significantly improved [71].

Another important direction of the development of density-based clustering is the extension with fuzzy-set theory. As previous discussions, a typical density-based clustering algorithm has a crisp-neighborhood model. A critical problem of this model is the difficulty in selecting appropriate parameters. In order to overcome this shortcoming, recently, building a soft model instead of the crisp model for neighborhoods has attracted a lot of attention. Fuzzy Joint Point (FJP) and its extension, Noise Robust FJP (NRFJP), introduced the concept of fuzzy-neighborhood into density-based clustering algorithms and partially solved this problem [22, 23]. Unfortunately, FJP-like method usually requires a very high time complexity. Another approach, Fuzzy-Neighborhood DBSCAN (FN-DBSCAN) provides a good balance between clustering efficiency and quality [24]. Since FN-DBSCAN is the preliminary of the proposed method in this dissertation, we will detail this algorithm in Section 3.1.2.

More interesting work for the improvements of density-based clustering algorithms can be found in the literature [68, 72, 73, 74].

2.2 Incremental Clustering

Another reasonable way of categorizing existing clustering techniques is to classify them into two types by the manner of processing data, i.e. batch clustering and incremental clustering (also called stream clustering or online clustering). The batch clustering requires that all the data are input to algorithm at once before carrying our clustering. On the other hand, the incremental clustering allows that data can be input and learned by the algorithm one by one. So, if the input dataset is changing over time, the incremental clustering is a much more efficient manner than the batch clustering to detect clusters in such datasets.

2.2.1 Classical Incremental Clustering Techniques

So far, the state of the art we summarized in the previous section are batch clustering techniques. It is also possible to extend the classical clustering with a capability of incremental clustering.

Incremental Partitonal Clustering

Recently, the implements of incremental k-means attracted a lot of attention. A well-known algorithm is STREAMLS [75], which partitions the input stream into chunks and clusters each chunk using a local search algorithm from [76]. Although it has been shown that STREAMLS is slower than BIRCH (a well-known incremental hierarchical clustering, see more discussions in the following part), it can usually obtain a better clustering quality than the later one. CluStream [77] is a stream clustering framework which allows the exploration of a stream over different time windows, hence providing a better understanding of the evolution of a stream [78]. The main idea of CluStream is to divide the clustering process into an online process that periodically stores summary statistics, and an offline process that uses only these summary statistics. N. Ailon et al proposed a modified k-means algorithm which can work well in an incremental mode [79]. In this work, they combined a deviation of the k-means++ algorithm which can output more than k centers, and a hierarchical clustering, so

that the modified algorithm has an impressive performance with limited memory. M. R. Ackermann et al proposed the StreamKM++ algorithm which consists of an adaptive, non-uniform sampling approach and a new data structure called coreset tree [80].

Incremental Hierarchical Clustering

Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) [81] incrementally and dynamically clusters based on a new data structure, called clustering feature (CF) tree which grows by aggregation, achieving a result with only one pass over the data, with time complexity of $O(n)$. Therefore, BIRCH can achieve an impressive efficiency for large-scale multi-dimensional datasets. However, BIRCH usually has a difficult in detecting arbitrary-shaped clusters [75]. S. Nassar proposed a new incremental hierarchical clustering based on a scheme to maintain data bubbles dynamically [82]. By using incremental data bubbles, a high-quality hierarchical clustering is quickly available at any point in time. In this new scheme, a quality measure for incremental data bubbles is used to identify data bubbles that do not compress well their underlying data points after certain insertions and deletions. Only these data bubbles are re-built using efficient split and merge operations. The Online Divisive-Agglomerative Clustering (ODAC) [83] uses a correlation-based dissimilarity measure between time series over a data stream and possesses an agglomerative phase to enhance a dynamic behavior capable of concept drift detection. Main features of ODAC include splitting and agglomerative criteria based on the diameters of existing clusters and supported by a significance level. At each new data point, only the leaves are updated, reducing computation of unneeded dissimilarities and speeding up the process every time the structure grows.

Incremental Density-based Clustering

DenStream [84] is an incremental version of DBSCAN. The concept, core-micro-cluster, is used to summarize the arbitrary-shaped clusters, while the potential-core-micro-cluster and outlier-micro-cluster structures are used to maintain and distinguish

the potential clusters and outliers. On the basis of these concepts, a pruning strategy is used to guarantee the precision of the weights of the micro-clusters with limited memory. Like DBSCAN working in the batch mode, DenStream can discover clusters of arbitrary shape in data streams, and it is insensitive to noise. D-Stream [85] is a density-based incremental clustering framework. It first maps each input data into a grid, then computes the density of each grid, and classifies the grids into clusters using a density-based approach. Due to the density-related concepts, D-Stream works better at finding arbitrary-shaped clusters than the previously introduced clustering framework, CluStream.

2.2.2 Incremental Clustering by Self-organizing Neural Networks

Although we have introduced some incremental extensions for each clustering category in the previous section, a more natural implementation is to use self-organizing neural networks which have been playing an important role in the field of unsupervised learning over the past few decades.

One of the most well-known techniques is self-organizing map (SOM) [86], also known as the Kohonen network. The original SOM has two layers: the input layer and the competitive layer. The competitive layer consists of a set of units (also called neurons or nodes) with lateral connections, which is usually constructed as a two-dimensional topological structure. A weight vector is assigned to each unit and is updated during training procedures according to the simple competitive learning (SCL) [87] strategy. When the training procedure is finished, SOM divides the input space into several regions, which can be considered that the input space is described by SOM as a Voronoi diagram, and each best matching unit (BMU) is the site of the corresponding Voronoi cell, indicating that SOM is suitable for clustering tasks. Another powerful feature of SOM is that high-dimensional data can be projected to a low-dimensional topological structure, which means that SOM can be applied to data visualization. Unfortunately, SOM has some drawbacks, one of which is its fixed

structure in its competitive layer. The size of the network must be predefined and is unchangeable during training procedures, which is similar to providing an integer number k as a parameter in the k-means clustering algorithm. In general, it is difficult to determine k without any prior knowledge of the given dataset [88]. In addition, the fixed structure limits SOM to the detection of clusters which are presented as complex shapes.

In order to overcome the drawback caused by the fixed structure, a series of 'growing-type' self-organizing neural networks have been proposed. These techniques usually have dynamic network structures.

Growing cell structures (GCS) [89] have a flexible network consisting of k-dimensional simplices. Each node is assigned a local variable, called a signal counter, which presents the local error of a neuron in the training procedure. GCS starts with a random k-dimensional simplex, e.g. the network is a triangular network if $k = 2$, and then, at timed intervals, new nodes will be inserted by splitting the *longest edge* emanating from the node with the maximum accumulated error. However, the topology dimensions in GCS must be kept strictly: when inserting a node, additional edges are also required to be added in order to maintain the topological structures. Further, another problem is that once a node is created in GCS, it cannot be removed.

Unlike SOM and GCS, the growing neural gas (GNG) method [90] has a more flexible network structure in which nodes can be added or removed, and there are no constrains for the topological structure. Actually, it can be considered as a combination of competitive Hebbian learning (CHL) [91] method (See more discussions in Section 6.2.1), neural gas (NG) [92] and the growing strategy of GCS. In the training procedure, GNG starts with two neurons. An edge will be created between two nodes with the highest activity if there is no edge between them. New nodes are inserted in the same way as that of GCS. The algorithm will stop if some predefined condition is met. Because of the presence of these important features, the GNG algorithm is considered to be suitable for the task of on-line learning.

With the increasing number and dimensions of data, the learning algorithms are required to efficiently deal with large number of signals. In general, there are two

main approaches to speed up the GNG algorithm. The first approach is to reduce the computational complexity of GNG, such as density-based growing neural gas (DB-GNG) [93] and the literature [94]. They usually establish a supporting structure, such as an R-tree, slim-tree or *kd*-tree, in order to reduce the cost of finding the nearest neuron. Another approach tries to modify the growing mechanism, such as fast autonomous growing neural gas (fAGNG) [95] and incremental growing neural gas (IGNG) [96]. The fAGNG algorithm performs the insertion of k neurons every λ signals. IGNG uses a distance-check strategy to insert neurons, which means that there is no necessity to calculate the local accumulated errors. Furthermore, it is not required to find the neurons with the maximum accumulated error when inserting neurons. These features make IGNG more efficient than the original GNG algorithm.

Considering that the given data distribution contains some noisy data, it will become a very difficult task for the methods mentioned above to learn such data distributions robustly. Unfortunately, few self-organizing neural networks were developed to address this problem. The self-organizing incremental neural network (SOINN) [97] was designed for the learning of non-stationary data distributions. SOINN has two layers in its architecture. The first layer performs like the GNG algorithm and the second layer is used to detect the potential low-density areas using the output of the first layer as its input. However, this two-layer architecture is sometimes too complicated for an on-line learning algorithm as users do not know when to stop the first layer and when to start the second layer. In addition, two layers mean that there are more parameters to be determined. In order to deal with noisy data, Robust Growing Neural Gas (RGNG) [98] extended the GNG algorithm with a noisy data resistant scheme. However, like the original GNG algorithm, RGNG also needs to calculate the local accumulated errors, which means that RGNG has a higher computational complexity than the IGNG like techniques.

Chapter 3

Landmark Fuzzy Neighborhood DBSCAN

In this chapter, we present our preliminary work, a fuzzy density-based clustering algorithm called Landmark Fuzzy Neighborhood DBSCAN (landmark FN-DBSCAN), which can provide a similar clustering quality to FN-DBSCAN, but only requires a linear time complexity to the size of input dataset.

3.1 Preliminaries

3.1.1 DBSCAN

DBSCAN assumes that the probability density of a small area in the feature space is uniform and the data density in each desired cluster is higher than the outside of the cluster. The density of noisy data is expected to be lower than that of normal data. To describe the density of data, a neighborhood for each data, called ε-neighborhood, is defined, and then, the main idea of DBSCAN is that for each *core-data* in a cluster, the number of data in its neighborhood has to exceed some threshold, and for each *border-data* in a cluster, there must exist at least one *core-data* such that the *border-data* are *density-reachable* from the *core-data* [21]. Now, we describe these concepts as follows.

Definition 1 (ε-*neighborhood* of a data) *Given a dataset D and a positive real number ε, the ε-neighborhood of a data x ($x \in D$), denoted as $N_\varepsilon(x)$, is a set of data defined by:*

$$N_\varepsilon(x) = \{x' \in D \mid dis\,(x', x) \leq \varepsilon\} \tag{3.1}$$

where $dis\,(x', x)$ represents the Euclidean distance between $x' = (x'_1, x'_2, ..., x'_n)$ and $x = (x_1, x_2, ..., x_n)$, i.e., $dis\,(x', x) = \sqrt{\sum_{i=1}^{n} (x'_i - x_i)^2}$. We say that x' is in the ε-neighborhood of x if $x' \in N_\varepsilon(x)$.

Definition 2 (core data) *Given a positive integer, $MinPts$, a data x ($x \in D$) is called a core data if:*

$$|N_\varepsilon(x)| \geq MinPts \tag{3.2}$$

where $|N_\varepsilon(x)|$, named the cardinality of x, is the number of data in the ε-neighborhood of x.

Definition 3 (directly density-reachable) *Given the two parameters, ε and $MinPts$, a data point x_i ($x \in D$) is directly denisty-reachable to the data point x_j, if x_j is a core data and $x_i \in N_\varepsilon(x_j)$.*

Fig. 3-1 (a) shows three data points lying on a part of data distribution, where x_1 is a border point, x_2 and x_3 are two *core points* (with the parameter setting as $MinPts = 6$). Directly density-reachable is symmetric for pairs of *core data* (Fig. 3-1 (b)). However, it is not symmetric if one core data and one border data are involved. Fig. 3-1 (c) shows the asymmetric case.

Definition 4 (density-reachable) *Given the two parameters, ε and $MinPts$, a data point x_i ($x \in D$) is denisty-reachable to the data point x_j, if there is a chain of data points $x_1, ..., x_n, x_1 = x_i, x_n = x_j$ such that x_{i+1} is directly-researchable from x_i.*

Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric since directly density-reachable is not symmetric[21]. Figure 3-2 (a) shows an example of an asymmetric case of density-reachable.

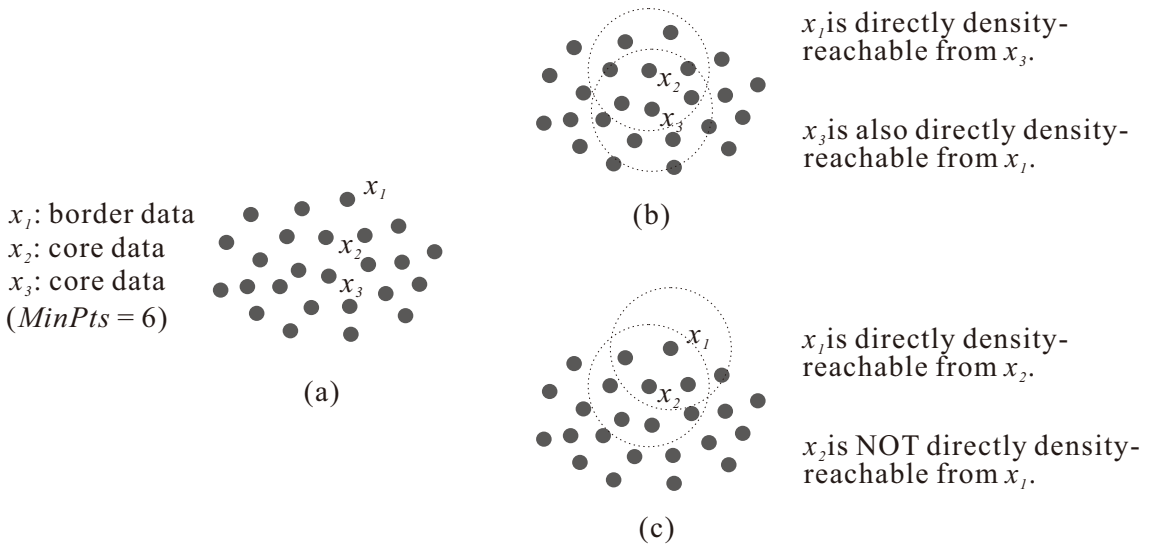


Figure 3-1: Two data points, x_1 and x_2 with $MinPts = 6$ (a). x_1 is directly density-reachable from x_2 (b), whereas x_2 is NOT directly density- reachable from x_1 (c).

Definition 5 (density-connected) *Given the two parameters, ε and $MinPts$, a data point x_i ($x \in D$) is denisty-connected to the data point x_j, if there is a data point x_k such that both x_i and x_j are density-reachable from x_k.*

Obviously, density-connectedness is a symmetric relation. Fig. 3-2 (b) shows a density-connected case.

On the basis of the above definitions, DBSCAN finds clusters in which each pair of data points are density-connected. We summarize the DBSCAN algorithm in Algorithm 4.

DBSCAN uses a distance-based strategy to describe the neighborhood of each data, i.e., the ε-neighborhood of a data (see Definition 1). Here, the parameter ε is used as a globally fixed value, which means the neighborhoods of all data have the same value of radius. And such a fixed value of radius results in a crisp boundary for the neighborhood. The neighborhood function of DBSCAN can be described by Equation (3.3).

$$f_x(x') = \begin{cases} 1, & \text{if } dis(x, x') \leq \varepsilon \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

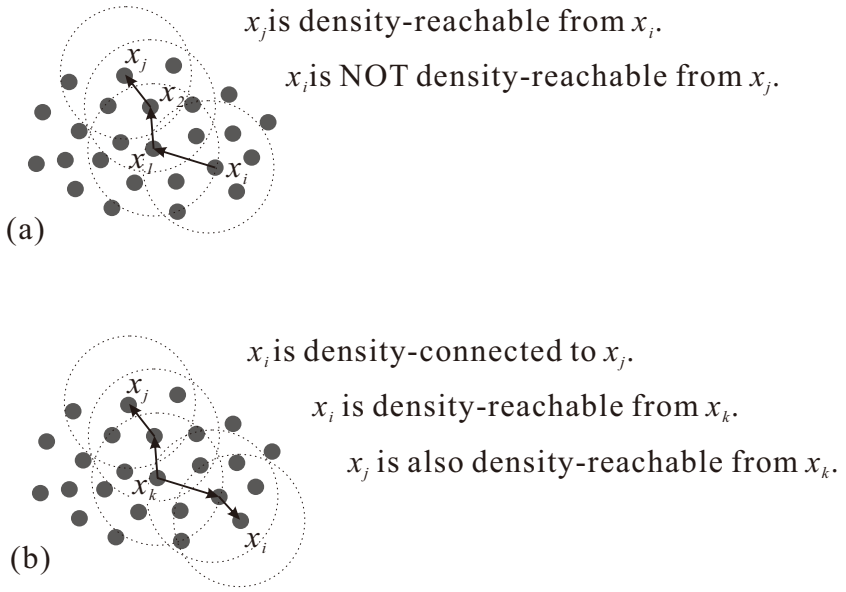


Figure 3-2: Two data points, x_i and x_j with $MinPts = 6$. (a) x_i is density- reachable from x_j, whereas x_j is NOT density- reachable from x_i. (b) x_i and x_j are density-connected

When a data is given, this model works well for determining whether another data in the dataset is a neighbor of the current one or not. However, since a key step of DBSCAN is to estimate the local density by calculating the *cardinality* of the ε-neighborhood of each data, it may not be always powerful to estimate the local density correctly, because two ε-neighborhoods of two different data may have the same *cardinality* according to Definition 2, although the density values are not the same. Fig. 3-3 shows an example of this situation. If we take a point of view of measuring the degree of the neighborhood membership for each pair of data, this phenomenon is caused by the same, crisp membership degree for all data in one neighborhood according to Equation (3.3). For example, consider that there are three data, x, x_1 and x_2, and we want to tell the difference of the membership degrees between two pairs of data, i. e. (x, x_1) and (x, x_2). Unfortunately, the membership degrees for (x, x_1) and (x, x_2) are the same given by the crisp neighborhood function (See Fig. 3-4(a)).

Algorithm 4 The DBSCAN Algorithm

Input: D, ε, $MinPts$
Output: $\mathcal{C}$ and "noise" mark

```
Mark all data as "unassigned";
C ← φ;
for all x in D do
  if x is marked as "unassigned" then
    N_ε(x) ← find the ε-neighborhood of x;
    card(x) ← Calculate the cardinality of x.
    if card(x) ≥ MinPts then
      Mark x as "assigned";
      Create a new cluster C, initialized by C ← {x};
      Create a queen S, initialized by S ← N_ε(x);
      for all p in S do
        C ← C ∪ {p};
        Mark p as "assigned";
        N_ε(p) ← find the ε-neighborhood of p;
        card(p) ← Calculate the cardinality of p.
        if card(p) ≥ MinPts then
          Add all data in N_ε(p) to the end of queen: S ← S ∪ N_ε(p);
        end if
      end for
      C ← C ∪ {C};
    end if
  end if
end for
Mark all remaining "unassigned" data as "noise";
```

3.1.2 Fuzzy Neighborhood DBSCAN (FN-DBSCAN)

To overcome the drawback mentioned in the previous section, recently, using more flexible, soft model to build the neighborhood has been developed. The Fuzzy Neighborhood DBSCAN (FN-DBSCAN) is a good extension of the original DBSCAN algorithm with the fuzzy set theory. The key idea of FN-DBSCAN is to use fuzzy neighborhood functions to define a new fuzzy neighborhood instead of the old crisp neighborhood. In the following part, two different fuzzy neighborhood functions, i.e. the linear one and the exponential one, are introduced.

Given two data x and x' ($x, x' \in D$), the linear neighborhood function is given by

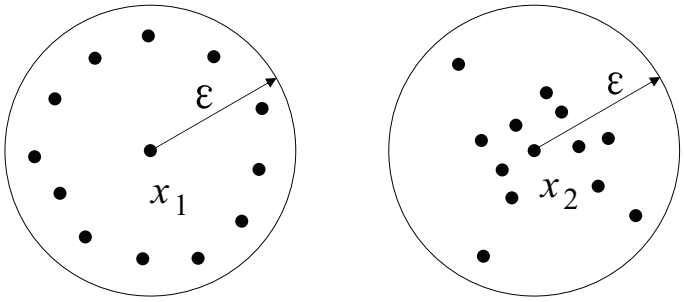


Figure 3-3: An example of using crisp neighborhood in DBSCAN. Data x_1 and x_2 have the same value of *cardinality* which is 12, but the values of the density are not the same.

Equation (3.4).

$$f_x(x') = \max\left\{1 - k\frac{dis(x,x')}{x^{max}}, 0\right\} \tag{3.4}$$

where k is a positive real number ($k > 0$) and x^{max} is the maximum distance of all pairs of data in D.

As expected, different membership degrees can be distinguished by applying linear neighborhood function. Fig. 3-4(b) shows that the membership degree of the data pair (x, x_1) is higher than that of the data pair (x, x_2).

Given two data x and x' ($x, x' \in D$), the exponential neighborhood function is given by Equation (3.5).

$$f_x(x') = \exp\left(-\left(k\frac{dis(x,x')}{x^{max}}\right)^2\right) \tag{3.5}$$

where k is a positive real number ($k > 0$) affecting the neighborhood radius and x^{max} is the maximum distance of all pairs of data in D.

Fuzzy Neighborhood DBSCAN (FN-DBSCAN) gives a generalized definition for describing a soft neighborhood.

Definition 6 (*fuzzy-neighborhood* of a data) *Given a dataset D and a positive real number ε_1, the fuzzy-neighborhood of a data x ($x \in D$), denoted as $N_f(x)$, is a set of data defined by:*

$$N_f(x) = \{x' \in D \mid f_x(x') \geq \varepsilon_1\} \tag{3.6}$$

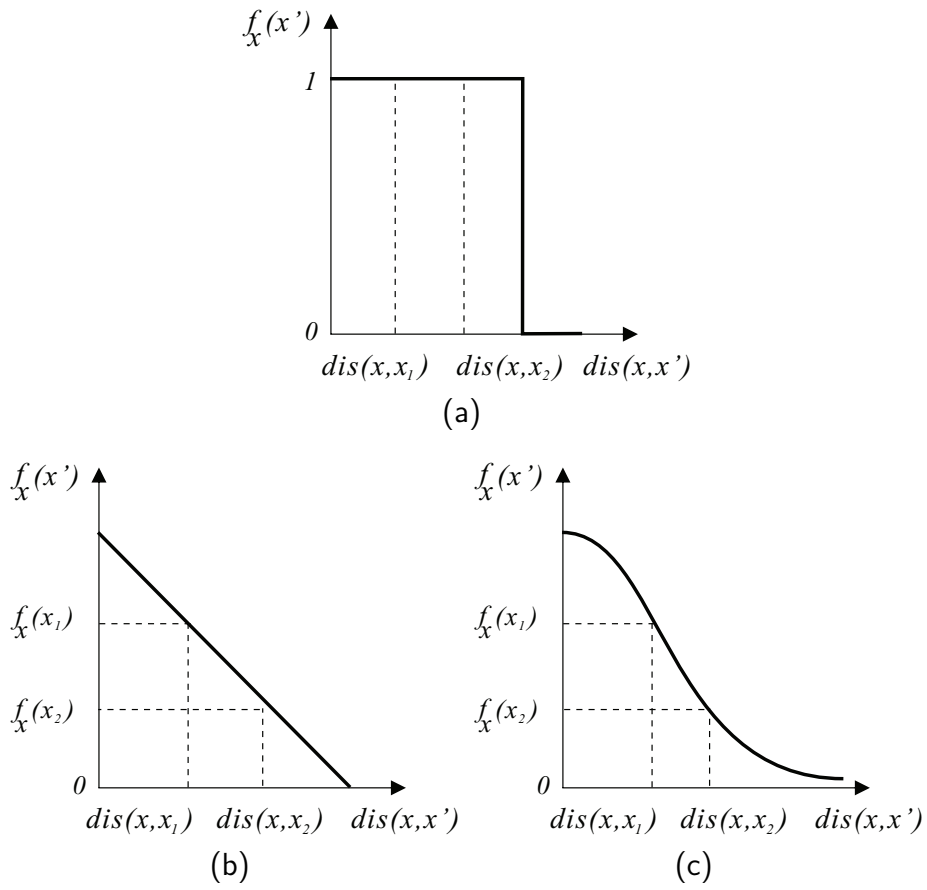


Figure 3-4: Different neighborhood functions. (a) The crisp neighborhood function. (b) The linear fuzzy neighborhood function. (c) The exponential fuzzy neighborhood function

where $f_x(x')$ is a generalized fuzzy neighborhood function which produces a membership degree for two data, x and x'. We say that x' is in the fuzzy neighborhood of x if $x' \in N_f(x)$.

Additionally, FN-DBASCAN modified the definition of the "core data".

Definition 7 (fuzzy core data) *Given a positive real number ε_2, a data x ($x \in D$) is called a $fuzzy$ core data if:*

$$|N_f(x)| \equiv \sum_{x' \in N_f(x)} f_x(x') \geq \varepsilon_2 \tag{3.7}$$

where $|N_f(x)|$, named $fuzzy$ cardinality, denotes the level of the density of the fuzzy

Algorithm 5 The FN-DBSCAN Algorithm

Input: D, ε_1, ε_2, f
Output: $\mathcal{C}$ and "noise" mark

```
Mark all data as "unassigned";
C ← φ;
for all x in D do
  if x is marked as "unassigned" then
    N_f(x) ← find the fuzzy neighborhood of x by applying f and ε1;
    card(x) ← Calculate the fuzzy cardinality of x.
    if card(x) ≥ ε2 then
      Mark x as "assigned";
      Create a new cluster C, initialized by C ← {x};
      Create a queen S, initialized by S ← N_f(x);
      for all p in S do
        C ← C ∪ {p};
        Mark p as "assigned";
        N_f(p) ← find the fuzzy neighborhood of p by applying f and ε1;
        card(p) ← Calculate the fuzzy cardinality of p.
        if card(p) ≥ ε2 then
          Add all data in N_f(p) to the end of queen: S ← S ∪ N_f(p);
        end if
      end for
      C ← C ∪ {C};
    end if
  end if
end for
Mark all remaining "unassigned" data as "noise";
```

neighborhood of x.

By replacing the Definition 1 and Definition 2 in the original DBSCAN algorithm with Definition 6 and Definition 7, respectively, we can transform the original, distance-based DBSCAN into the level-based FN-DBSCAN. Thus in the previous example shown in Fig 3-3, the *fuzzy cardinality* of x_1 and x_2 are not the same according to Definition 7.

With specifying parameters ε_1, ε_2 and the fuzzy neighborhood function f, the FN-DBSCAN algorithm can be implemented as Algorithm 5.

We see the FN-DBSCAN algorithm is very close to the original DBSCAN algorithm. Actually, if FN-DBSCAN uses the same way of defining neighborhoods and calculating the cardinalities, FN-DBSCAN will become DBSCAN. Besides, we easily

know the time complexity of FN-DBSCAN is $O(n^2)$, which is the same as DBSCAN.

3.2 Landmark Fuzzy Neighborhood DBSCAN

Despite the fact that the FN-DBSCAN made the density-based clustering more robust, the standard FN-DBSCAN algorithm has a time complexity of $O(n^2)$, indicating an expensive time complexity for large-scale datasets.

3.2.1 Landmark Generation

In order to reduce the expensive cost on processing the dataset directly, the input dataset is divided into several subsets with smaller sizes. In this procedure, a set of data are selected and further processed as "landmarks" and each landmark is used to represent one subset. Here we give several related concepts in this procedure.

Definition 8 (landmark) *Given a dataset D as an n-by-m matrix, where n is the number of data and m is dimensionality of data, a landmark, denoted as l, is a triplet:*

$$l = \left\langle \mathbf{V}, N_f^L(l), \mu \right\rangle \tag{3.8}$$

where $\mathbf{V}$ is an m-dimensional vector corresponding to a data in D (determined by Algorithm 6), $N_f^L(l)$ is a subset of D, containing all the data in the fuzzy neighborhood of l (See Definition 9) and μ is a positive real number called the membership level of l.

With the property $\mathbf{V}$, a landmark is able to compare the membership degree with a data or another landmark. In order to measure the membership degree in such two cases, two variants of exponential fuzzy neighborhood functions are used.

First, given a landmark l and a data x ($x \in D$) for example, we measure the membership degree between l and x by, say, Equation (3.9),

$$f_l(x) = \exp\left(-\left(r \cdot k \cdot \frac{dis(\mathbf{V}, x)}{\Delta d^{max}}\right)^2\right) \tag{3.9}$$

where r, k are positive real numbers and Δd^{max} is the maximum distance between $\mathbf{V}$ and all the other data in D.

Second, as for the membership degree between two landmarks l_1 and l_2, we define the membership function as, say, Equation (3.10),

$$f_{l_1}(l_2) = \exp\left(-\left(k \cdot \frac{dis(\mathbf{V_1}, \mathbf{V_2})}{\Delta d^{max}}\right)^2\right) \tag{3.10}$$

where k is a positive real number and Δd^{max} is the maximum distance between $\mathbf{V_1}$ and all the other landmarks.

Taking Equation (3.9) for example, we describe how the above two exponential fuzzy neighborhood functions were designed. Given a data x ($x \in D$) and a landmark l, the function $f_l(x) = \exp(-dis(\mathbf{V}, x))$ is used to make the membership degree fixed in the range of $(0, 1]$, where $f_l(x)$ approaches to 0 when the distance between l and x approaches to infinity. In addition, we want the membership degree to be more sensitive to the small distance between l and x, which means two close points can get a greater value of the membership degree. For this purpose, the function was transformed to $f_l(x) = \exp(-(dis(\mathbf{V}, x))^2)$. In practice, when the dataset D and a landmark l are given, the maximal distance between $\mathbf{V}$ and all the data in D, denoted as Δd^{max}, can be considered as a constant. So we further transformed the function to $f_l(x) = \exp(-(\frac{dis(\mathbf{V},x)}{\Delta d^{max}})^2)$. Finally, the parameters k and r are used to adjust the radius of the neighborhood, i.e., the function was finally transformed to $f_l(x) = \exp\left(-\left(r \cdot k \cdot \frac{dis(\mathbf{V},x)}{\Delta d^{max}}\right)^2\right)$.

Here, we make a further discussion on the parameters k and r in Equation (3.9) and Equation (3.10). Comparing these two equations, we see that both of them use parameter k, and here k is used as the same value. The reason why two parameters k and r are used in Equation (3.9) is that it is necessary to emphasize that those two equations are used in different steps. Equation (3.9) is used only in landmark generating procedure (Step (1), see Algorithm 6). The other Equation is only used in Step (2) (See Algorithm 7). Since these two steps have different purposes, the radii of the neighborhood may be different. Hence, the parameter r used in Equation (3.9)

Algorithm 6 The Landmark Generation Algorithm

Input: D, r, k, ε_1
Output: Ł

Ł $\leftarrow \emptyset$;
for all x in X **do**
  Find a landmark, l, such that:
  $l \leftarrow \arg\min_{l \in \text{Ł}}\{d(\mathbf{V}(l), x)\}$;
  $u \leftarrow \exp\left(-\left(r \cdot k \cdot \frac{dis(l.V,x)}{\Delta d^{max}}\right)^2\right)$;
  if Ł $= \emptyset$ or $u(l) < \varepsilon_1$ **then**
    $\mathbf{V}(l) \leftarrow x$; $N(l) \leftarrow \{x\}$; $\mu(l) \leftarrow 1$;
    $l \leftarrow (\mathbf{V}(l), N(l), \mu(l))$;
    Ł $\leftarrow$ Ł $\cup \{l\}$;
  else
    $N(l) \leftarrow N(l) \cup \{x\}$;
    $\mu(l) \leftarrow \mu(l) + u(l)$;
  end if
end for

offers an additional adjustment for the neighborhood radius in Step (1).

Definition 9 (fuzzy-neighborhood of a landmark) *Given a set D, where D can be a set of data or a set of landmarks, and a positive real number ε_1, the fuzzy-neighborhood of a landmark l, denoted as $N_f^L(l)$, is the set of data or the set of landmarks defined by:*

$$N_f^L(l) = \{x \in D \mid f_l(x) \geq \varepsilon_1\} \tag{3.11}$$

where $f_l(x)$ can be Equation (3.9) or Equation (3.10). Similar to Definition 6, we say that x is in the fuzzy neighborhood of landmark l, if x belongs to $N_f^L(l)$.

The last property of a landmark, the membership level, μ, can be calculated by the following equation:

$$\mu = \sum_{x \in N_f^L(l)} f_l(x) \tag{3.12}$$

With the above concepts we give a way of generating landmarks in Algorithm 6.

From Algorithm 6, we see landmarks are generated dynamically during the algorithm's execution and all data in the dataset can and only can belong to one landmark. An example of generating landmarks on an artificial dataset is shown in Fig. 3-5. All

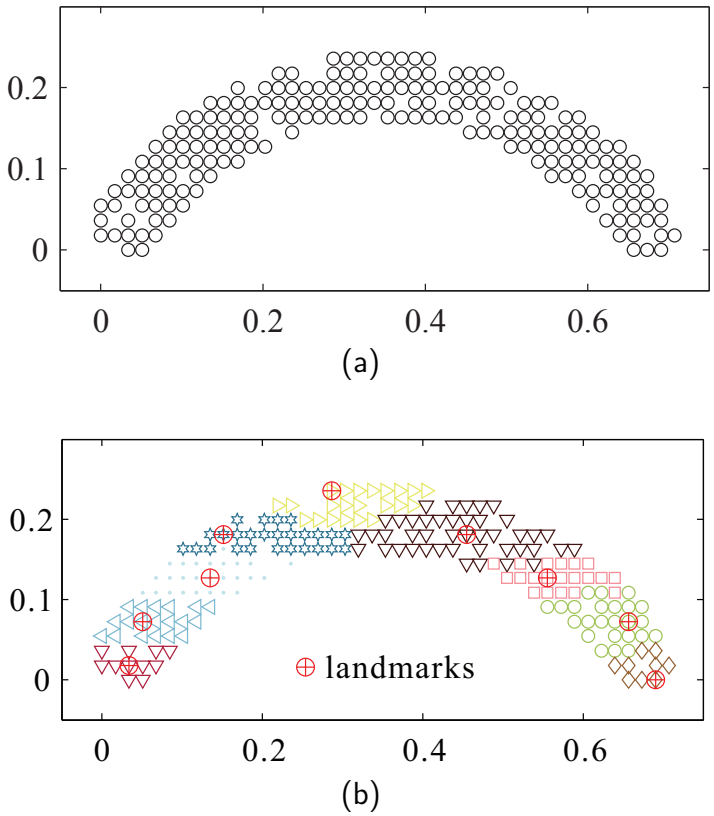


Figure 3-5: Example of generated landmarks on "arc" shaped dataset. (a) The "arc" dataset. (b) Generated landmarks

data are labelled by each corresponding landmark.

3.2.2 A Modified Version of FN-DBSCAN

Since the output of Algorithm 6 is a set of landmarks, which is not acceptable by the original FN-DBSCAN algorithm, we modify the standard FN-DBSCAN so that the modified version can process landmarks.

First, we give a modified method to calculate the cardinality of the neighborhood of a landmark. Consider a set of landmarks, L, where $l = \langle \mathbf{V}, N_f^L(l), \mu \rangle \in L$. The cardinality of the neighborhood of l can be calculated by the following equation:

$$card(l) = \sum_{l' \in N_f^L(l)} l'.\mu \tag{3.13}$$

Next, we show the details of this modified version in Algorithm 7.

Algorithm 7 A Modified FN-DBSCAN Algorithm

Input: L, ε_1, ε_2, k
Output: $\mathcal{C}$ and "noise" mark

```
Mark all data as "unassigned";
C ← φ;
for all l in L do
  if l is marked as "unassigned" then
    N_f^L(l) ← find the fuzzy neighborhood of l by applying Equation (3.10) to
    Definition 9 wrt. ε1 and k;
    card(l) ← Calculate the cardinality of l by Equation (3.13);
    if card(l) ≥ ε2 then
      Mark l as "assigned";
      Create a new cluster C, initialized by C ← {l};
      Create a queen S, initialized by S ← N_f^L(l);
      for all p in S do
        C ← C ∪ {p};
        Mark p as "assigned";
        N_f^L(p) ← find fuzzy neighborhood of p by applying Equation (3.10) to
        Definition 9 wrt. ε1 and k;
        card(p) ← Calculate the cardinality of p by Equation (3.13);
        if card(p) ≥ ε2 then
          Add all data in N_f^L(p) to the end of the queen: S ← S ∪ N_f^L(p);
        end if
      end for
      C ← C ∪ {C};
    end if
  end if
end for
Mark all remaining "unassigned" data as "noise";
```

3.2.3 The Landmark Fuzzy Neighborhood DBSCAN Algorithm

The landmark FN-DBSCAN algorithm is a three-phase clustering algorithm described as follows.

1. Divide a dataset into several subsets represented by generated "landmarks" according to Algorithm 6.

2. Execute a modified version of FN-DBSCAN (Algorithm 7) on the generated landmark set and output the index of landmarks.

3. Label data according to the cluster-indices of the landmarks.

Now, we summarize our algorithm based on the above discussions in Algorithm 8.

Algorithm 8 The Landmark FN-DBSCAN Algorithm

Input: D, ε_1, ε_2, r, k
Output: P

1: $L \leftarrow$ Execute the Landmark Generation algorithm (Algorithm 6) wrt. D, r, k, ε_1;
2: $C \leftarrow$ Execute the modified FN-DBSCAN algorithm (Algorithm 7) wrt. $L, \varepsilon_1, \varepsilon_2, k$;
3: $P \leftarrow$ Label D by C.

3.2.4 Complexity Analysis

Theorem 1 *The time complexity of landmark FN-DBSCAN is $O(mn + m^2)$, where n is number of data and m is the number of generated landmarks.*

In Step (1), the algorithm needs to scan the dataset once, which takes $O(n)$ time complexity. And in each loop, to find the landmark which has the minimum distance to the current data, it is expected to take an O(1/2*mn) time complexity in total. Calculating Δd^{max} takes $O(mn)$. So, the time complexity of Step (1) is $O(mn)$. In Step (2), it has the same time complexity as FN-DBSCAN, which is $O(m^2)$. Obviously, Step (3) costs $O(n)$ time complexity. Therefore, the time complexity of landmark FN-DBSCAN is $O(mn + m^2 + n) = O(mn + m^2)$

However, in practice, the number of generated landmarks is much less than the number of data, i.e., $m \ll n$. In this case, the time complexity of landmark FN-DBSCAN reduces to $O(n)$, which indicates that it is suitable to large datasets.

Theorem 2 *The space complexity of landmark FN-DBSCAN is $O(n + m)$, where n is the number of data and m is the number of generated landmarks.*

In Step (1), the algorithm requires the space complexity of $O(n)$ to store the dataset and needs $O(m)$ to store the generated landmarks. Besides, it also needs to store the indexes of all data in neighborhoods of landmarks, which is $O(n)$ in total. In Step (2), it requires the same space complexity as the FN-DBSCAN, which is $O(m)$. Step (3) needs no more extra space. Therefore, the space complexity is $O(n + m + n + m) = O(n + m)$.

Similar to the time complexity, the space complexity will reduce to $O(n)$, if $m \ll n$.

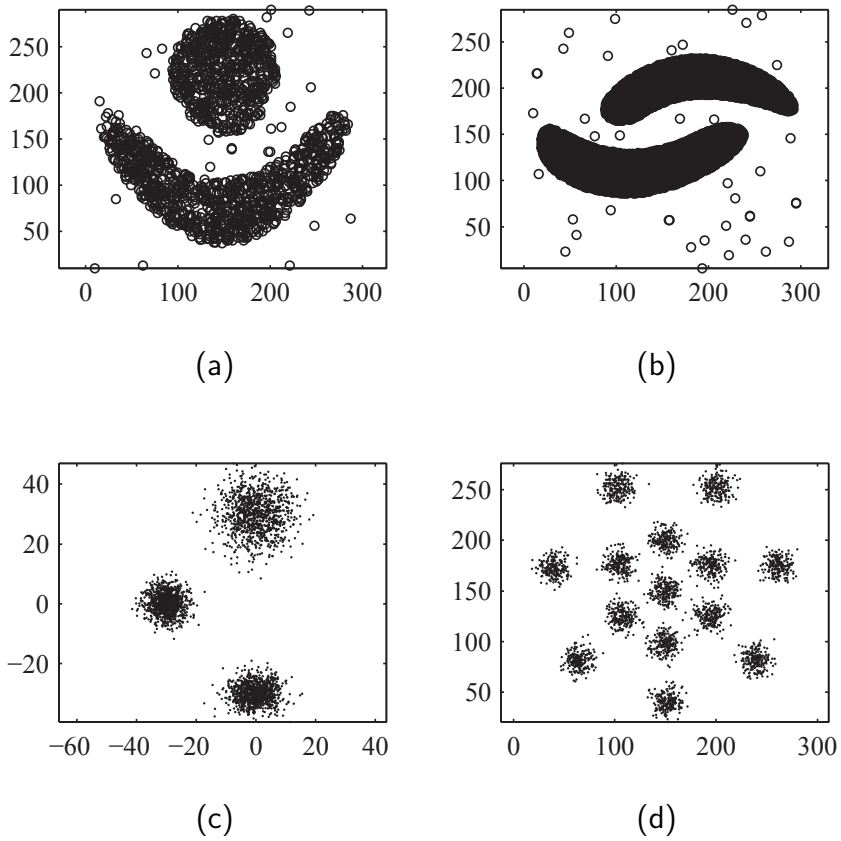


Figure 3-6: Synthetic datasets. (a) Anchor dataset. (b) Banana dataset. (c) Gaussian dataset (3 clusters). (d) Gaussian dataset (14 clusters).

3.3 Experimental Results

3.3.1 Experiments on Clustering Quality and Efficiency Comparing with FN-DBSCAN

In this section, both efficiency and clustering quality of the landmark FN-DBSCAN are tested for comparing with the original FN-DBCAN algorithm. Four synthetic datasets and two real world datasets are used. The details of these datasets are shown in Table 3.1 and four synthetic datasets are illustrated in Fig. 3-6.

Table 3.1: datasets Used in Experiments.

name	**Size**	**♯dimensions**	**♯clusters**	**noise**
Anchor	28196	2	2	yes
Banana	16532	2	2	yes
Gaussian (3 clusters)	30000	2	3	no
Gaussian (14 clusters)	28000	2	15	no
Letter	20000	16	26	no
Pendigits	10992	16	10	no

The symbol ♯ denotes "the number of".

The quality of clustering results is evaluated by comparing with the true partition (gold standard). A most well-known method, Rand-Index [99], is used in the following experiments, which are described briefly as follows. We briefly introduce the Rand-Index method in Appendix A.3. A higher Rand-Index value means a smaller distortion.

All the experiments were conducted by the Intel Core i5 2.67GHz CPU (with two of four threads available) with 2GB RAM running on Windows XP operating system. Both of the FN-DBSCAN algorithm and the landmark FN-DBSCAN algorithm were realized in Matlab language and were executed on the platform of Matlab 2011b (32-bit version).

In order to provide a fair comparison between FN-DBSCAN and the landmark FN-DBSCAN, the parameters were determined by the following rule. Actually, the authors of FN-DBSCAN suggested a general way of selecting parameters ε_1 and k,

i.e., choosing greater values of parameters ε_1 and k if datasets have strong density and choosing smaller values of these two parameters if datasets have low density. The parameter ε_2 is related to noisy data detection, greater value being suggested to deal with a large number of noisy data [24]. Since both of these two algorithms use parameters ε_1, ε_2 and k, we first determined the best combination of parameters ε_1, ε_2 and k for FN-DBSCAN when a dataset was given, and then, applied the same values of these parameters to the landmark FN-DBSCAN. As the landmark FN-DBSCAN uses an additional parameter r, it can provide a satisfactory clustering result by selecting an appropriate value of parameter r. We tested the landmark FN-DBSCAN algorithm with different values of r in the following experiments.

In addition, because FN-DBSCAN requires the input data be normalized before carrying out clustering, we normalized all data so that these two algorithms can accept the same values of parameters of ε_1, ε_2 and k. Here, the normalizing method used in FN-DBSCAN was used again, as summarized in the Appendix A.1.

In the following experiments, datasets with several different sizes were prepared, but they shared the same data distributions as their original ones. Besides, correct answers for each dataset (including each size for each dataset) were pre-prepared and then used in Rand-Index calculation to verify the results given by the landmark FN-DBSCAN and FN-DBSCAN.

The first two datasets, Anchor dataset and Banana dataset, were used to test whether each algorithm had a capability of detecting clusters in arbitrary shape. Besides, both of them had some noisy data.

1. The Anchor dataset, illustrated in Fig 3-6(a), consists of two clusters including 28196 data with a shape like an "anchor". The clustering result is shown in Table 3.2. Generally speaking, both the landmark FN-DBSCAN and FN-DBSCAN could provide a good quality on this dataset, but the former one was much faster than the latter one. When the size of dataset was 2500, it saved 93% time of FN-DBSCAN and even provided a slight better quality ($r = 1.5$). With increasing the number of data, the time cost by the landmark FN-DBSCAN was less than 2% of the time cost by FN-DBSCAN.

2. Banana dataset has two banana shaped clusters, illustrated in Fig. 3-6(b). Experimental results are shown in Table 3.3. When the size was 1500, the landmark FN-DBSCAN was 5 times faster than FN-DBSCAN ($r = 1.8$). But when the dataset size was increased to 16532, the landmark FN-DBSCAN became 40 times faster than FN-DBSCAN, providing a perfect clustering quality on this dataset.

The following two datasets are Gaussian datasets. Usually, density-based clustering algorithms are weak in dealing with Gaussian distributed data. But by using fuzzy neighborhood instead of classical crisp neighborhood, FN-DBSCAN can be expected to have a better capability of dealing with such datasets. So the following two datasets are used to test the proposed algorithm in this situation.

1. Fig. 3-6(c) shows a dataset containing 3 clusters each obeying a Gaussian distribution. The experimental result is summarized in Table 3.4. We see both of the landmark FN-DBSCAN and FN-DBSCAN got a good result on this dataset. Again, the landmark FN-DBSCAN was much more efficient than FN-DBSCAN. When dataset size was 30000, the landmark FN-DBSCAN was over 190 times faster than FN-DBSCAN, but provided the same quality ($r = 3.0$).

2. The result was similar when the number of clusters was increased to 14 (see Fig. 3-6(d)), which is shown in Table 3.5. The landmark FN-DBSCAN was still faster than FN-DBSCAN and provided similar quality.

In the following experiments, we observed that a cluster represented in a convex-shape could be represented by one landmark, if each cluster in the dataset was not overlapped with another cluster. The Gaussian datasets (3000 data, 3 clusters) were used in this experiment. Smaller values of r and ε_1 were used to generate smaller number of landmarks. The Fig. 3-7 shows two cases of the generated landmarks by using two different settings of the parameters. We can see that many landmarks

were generated to represent one cluster in Fig. 3-7(a), but only three landmarks were generated and each landmark represents one cluster in Fig. 3-7(b). Besides, we also notice that both of these two parameter settings could obtain good results of clustering, which were 0.9953 and 0.9969 in the Rand-index value, respectively. Since using smaller values of r and ε_1 can generate smaller number of landmarks, it can be expected to be faster than using greater values. In this experiment, the execution time was 0.09 when using smaller values, and the execution time was 0.13 when using greater values. (See the last paragraph of this section and Section 3.3.4 for more detailed discussions about selecting the parameter r).

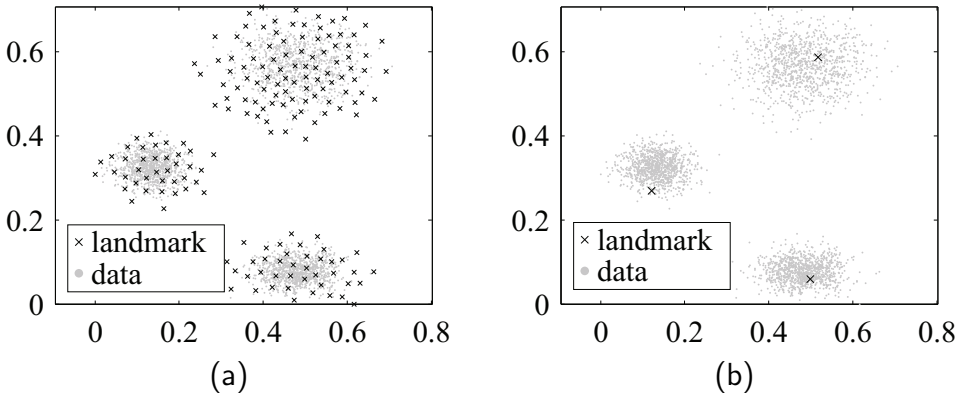


Figure 3-7: The landmarks generated by different parameter settings when using the landmark FN-DBSCAN to discover convex-shaped clusters. (a) 178 landmarks were generated when $\varepsilon_1 = 0.7, \varepsilon_2 = 10, k = 10, r = 1.5$. The execution time was 0.13 and the value of Rand-Index was 0.9953.(b) 3 landmarks were generated when $\varepsilon_1 = 0.05, \varepsilon_2 = 10, k = 10, r = 0.3$. The execution time was 0.09 and the value of Rand-Index was 0.9969.

The last two datasets were selected from UCI Machine Learning Repository, which are Letter dataset and Pendigits dataset (see the website: http://archive.ics.uci.edu/ml/). The result of using the landmark FN-DBSCAN and FN-DBSCAN are summarized in Table 3.6 and Table 3.7, respectively. The results similar to the previous experiments were obtained. The landmark FN-DBSCAN was over 12 times faster than FN-DBSCAN for 20000 size Letter dataset ($r = 0.7$) and over 267 times faster for 10992 Pendigits dataset ($r = 0.2$). However, we notice that FN-DBSCAN provided a better quality than the landmark FN-DBSCAN on the Pendigist dataset, but they

are very close. And the landmark FN-DBSCAN got a better result on the Letter dataset.

Now, we sum up the results influenced by the parameter r. For the datasets in which clusters represent complex shapes such as the Anchor dataset and the Banana dataset, the parameter r should be set as $r > 1$ in order to generate a sufficient number of landmarks to describe the data distributions well. We notice that although there are some exceptions, in general, with a greater value of r, we can obtain a better clustering quality than a smaller value of r, if the other parameter values are fixed. This is because the greater number of landmarks can describe the data distributions more precisely than the smaller number of landmarks. However, the value of r strongly influences the efficiency. A greater value of r costs much time than a smaller value of r, because a larger number of landmarks are generated and calculated if r is large. On the other hand, for the datasets where clusters are represented in convex-shapes such as Gaussian, Letter and Pendigits datasets, there are two strategies to decide the parameter r. One is using $r > 1$ to generate many landmarks like the previous discussions, and another way is to use small values of r ($r < 1$) and ε_1 to generate small number of landmarks. In this situation, one landmark can represent one cluster in the dataset like what Fig. 3-7 shows.

3.3.2 Experiments on Generating Landmarks with Different Input Orders of Data

Since the landmarks are generated dynamically according to the input datasets, it is possible that the input order of data may affect the generated landmarks. So, in this section, we verify whether the input order of data will influence the generated landmarks and whether it will influence the final clustering result.

The Banana dataset (with 1500 data in total) was used in this experiment. Two completely random input orders of this dataset were prepared. Since the purpose was to test the robustness in the cases of different input orders, the parameters were required to be kept as the same value in all the cases. The parameters were selected

Table 3.2: Anchor dataset.

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN Execution Time (s)** r **value**					**FET**
	1.5	1.8	2.0	2.5	3.0		1.5	1.8	2.0	2.5	3.0	
2500	0.9995	0.9713	0.9995	0.9987	0.9987	0.9987	0.06	0.08	0.08	0.09	0.13	0.86
5000	0.9895	0.9825	0.9819	0.9504	0.9924	0.9920	0.19	0.20	0.22	0.30	0.34	4.36
7500	1.0000	0.9998	0.9998	0.9998	0.9998	0.9998	0.45	0.59	0.67	0.94	1.23	7.75
10000	0.9998	0.9999	0.9999	0.9996	0.9996	0.9996	0.59	0.72	0.84	1.16	1.08	12.67
12500	0.9897	0.9996	0.9996	0.9993	0.9994	0.9993	0.75	0.58	0.66	1.55	1.45	12.05
15000	0.9973	0.9997	0.9998	0.9997	0.9996	0.9996	0.55	0.69	0.75	1.09	1.44	16.59
17500	0.9980	0.9999	0.9999	1.0000	0.9999	0.9999	0.84	1.13	1.31	1.83	2.63	22.28
20000	0.9835	0.9999	0.9999	0.9998	0.9999	0.9996	1.03	1.34	1.69	2.36	3.91	86.91
22500	0.9989	0.9999	0.9999	0.9999	0.9998	0.9997	1.14	1.61	1.95	2.67	4.34	93.50
28196	0.9998	0.9971	0.9998	0.9997	0.9998	0.9996	1.69	2.28	2.70	4.03	6.00	111.89

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time(s).

Parameters: 2500: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 10$; 5000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 10$;
7500: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$; 10000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$;
12500: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$; 15000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$;
17500: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 25$; 20000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 25$;
22500: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 25$; 28196: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 25$.

as: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$ and $r = 2$. The results of applying different input orders are shown in Fig. 3-8, where each result is shown by two figures. The left figure shows all the generated landmarks and the clustering result. The landmarks are marked by red-circle markers and the data in different clusters are depicted in different colors and shapes. The right figure shows a magnified part of data in the left figure (the data within the rectangle), which further illustrates the neighborhood of each landmark (shown by the colored data in the fuzzy neighborhood of a landmark) and the membership level of each landmark (shown by the number besides the corresponding landmark).

We notice that by applying two different input orders of data, two groups of landmarks were generated in different positions, neighbors and the membership levels. However, the final clustering results were almost the same. When we used Rand-Index to measure the clustering quality, both of these two results obtained 1.0 Rand-Index

Table 3.3: Banana dataset.

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN execution time(s)** r **value**					**FET**
	1.5	1.8	2.0	2.5	3.0		1.5	1.8	2.0	2.5	3.0	
1500	0.9751	1.0000	1.0000	1.0000	1.0000	1.0000	0.05	0.05	0.05	0.08	0.08	0.25
3000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.08	0.08	0.16	0.11	0.13	1.03
4500	0.9934	0.9998	0.9998	0.9998	0.9998	0.9998	0.17	0.17	0.20	0.28	0.36	3.77
6000	0.9994	0.9998	0.9994	0.9998	0.9998	0.9994	0.14	0.19	0.23	0.17	0.33	5.77
7500	0.9097	1.0000	1.0000	1.0000	0.9999	0.9999	0.25	0.31	0.34	0.45	0.59	7.95
9000	0.9999	1.0000	0.9999	1.0000	1.0000	0.9999	0.31	0.36	0.39	0.52	0.69	12.09
10500	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.27	0.30	0.47	0.58	0.48	15.31
12000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.44	0.39	0.67	0.88	1.22	20.89
13500	0.9951	1.0000	1.0000	1.0000	1.0000	1.0000	0.39	0.42	0.47	0.63	0.77	13.97
16532	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.48	0.56	0.66	0.84	1.11	19.23

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time (s).

Parameters: 1500: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 3000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
4500: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 6000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
7500: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 12$; 9000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 12$;
10500: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 12$; 12000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 15$;
13500: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 15$; 16532: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 15$.

value, which implies the landmark FN-DBSCAN algorithm got the correct answer in the cases of applying two completely random input orders.

We further investigated why the proposed method could get the same result with different generated landmarks. From Algorithm 8, we see the second step processes only landmarks. In order to get the same result with the one obtained by directly processing the input data, the generated landmarks must be able to represent the input dataset. The key elements to represent a dataset is the data density and the data distribution. So the following experiment was designed to test whether the landmarks generated by applying different input orders could give similar representation of the data density and data distribution.

First, we selected a small part of data of the upper cluster (the data within the rectangle in the left figure in Fig. 3-8) and we also recorded the corresponding landmarks generated by this part of data (shown in the right figure). We calculated

Table 3.4: Gusssian dataset (3 Clusters).

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN Execution Time (s)** r **value**					**FET**
	1.5	1.8	2.0	2.5	3.0		1.5	1.8	2.0	2.5	3.0	
3000	0.9953	0.9976	0.9980	0.9982	0.9987	0.9991	0.13	0.13	0.16	0.22	0.27	2.00
6000	0.9669	0.9982	0.9985	0.9990	0.9994	0.9996	0.20	0.25	0.30	0.36	0.45	6.08
9000	0.9921	0.9861	0.9814	0.9999	0.9999	0.9999	0.20	0.20	0.23	0.28	0.55	12.34
12000	0.9926	0.9928	0.9969	0.9962	0.9995	0.9996	0.25	0.28	0.31	0.36	0.44	20.55
15000	0.9983	0.9984	0.9995	0.9995	0.9995	0.9996	0.34	0.36	0.41	0.45	0.58	24.05
18000	0.9924	0.9990	0.9994	0.9996	0.9996	0.9997	0.39	0.44	0.45	0.56	0.67	38.56
21000	0.9976	0.9995	0.9996	0.9995	0.9998	0.9998	0.45	0.52	0.56	0.67	0.80	47.28
24000	0.9955	0.9998	0.9998	0.9999	0.9999	0.9999	0.53	0.59	0.66	0.75	0.92	127.11
27000	0.9943	0.9973	1.0000	1.0000	1.0000	1.0000	0.58	0.66	0.70	0.88	1.06	137.14
30000	0.9992	0.9998	0.9998	0.9998	0.9999	0.9999	0.64	0.72	0.81	0.97	1.14	218.30

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time (s).

Parameters: 3000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 6000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
9000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 12000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
15000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 18000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
21000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 24000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$;
27000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$; 30000: $\varepsilon_1 = 0.7$, $\varepsilon_2 = 10$, $k = 10$.

the sums of the membership levels of all the landmarks, and we noticed that such sums with different input orders were very close. In this specific case, the sum of membership levels with the order_1 was 46.65 and the sum with the order_2 was 46.23. And when we continued to calculate such sums for the landmarks generated by the data in each cluster, we got similar results: the sum for the upper cluster with the order_1 was 734.14 and the sum was 731.52 with the order_2. The sum for the lower cluster with the order_1 was 606.44 and the sum with the order_2 was 608.23. Since the sum of membership levels is related to the data density, we can say the input orders of data did not influence the representation of data density locally or globally.

The second step of the proposed method calls a modified version of the FN-DBSCAN algorithm (See Algorithm 7) where landmarks are considered as the input data. Since Algorithm 7 is also a density-based algorithm, the two concepts, *density-*

Table 3.5: Gusssian dataset (14 Clusters).

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN Execution time (s)** r **value**					**FET**
	1.2	1.4	1.8	2.0	2.5		1.2	1.4	1.8	2.0	2.5	
2800	0.9576	0.9848	0.9992	0.9991	0.9992	0.9694	0.11	0.13	0.17	0.19	0.25	1.61
5600	0.9564	0.9815	0.9981	0.9987	0.9990	0.9892	0.17	0.20	0.25	0.30	0.41	4.16
8400	0.9572	0.9796	0.9691	0.9691	0.9691	0.9388	0.44	0.45	0.61	0.73	1.11	9.77
11200	0.9173	0.9844	0.9970	0.9981	0.9886	0.9888	0.56	1.02	1.42	1.59	2.33	16.45
14000	0.9466	0.9876	0.9984	0.9988	0.9989	0.9992	0.56	0.67	1.00	1.09	1.89	82.08
16800	0.9429	0.9880	0.9988	0.9991	0.9792	0.9793	0.73	0.89	1.25	1.55	2.09	25.91
19600	0.9393	0.9880	0.9984	0.9989	0.9992	0.9994	1.00	1.30	1.70	2.13	2.98	90.34
22400	0.9435	0.9872	0.9984	0.9988	0.9991	0.9993	1.33	1.55	2.20	3.11	3.81	106.91
25200	0.9452	0.9887	0.9984	0.9987	0.9990	0.9992	1.72	1.89	2.88	3.36	4.83	113.58
28000	0.9402	0.9870	0.9888	0.9789	0.9791	0.9691	1.88	2.30	3.88	4.02	5.72	184.05

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time.

Parameters: 2800: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 15$; 5600: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$;
8400: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 20$; 11200: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 20$;
14000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 28$; 16800: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 30$;
19600: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 33$; 22400: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 35$;
25200: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 38$; 28000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 20$, $k = 38$.

reachable and *density-connected*, are directly related to the data distribution (See [21] and [24] for more detailed discussions). We notice that for any landmark in a

Table 3.6: Letter dataset.

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN Execution time(s)** r **value**					**FET**
	0.5	0.6	0.7	0.8	0.9		0.5	0.6	0.7	0.8	0.9	
2000	0.9612	0.9623	0.9623	0.9622	0.9621	0.9620	0.45	0.84	0.98	1.27	3.23	1.69
4000	0.9611	0.9620	0.9623	0.9624	0.9623	0.9528	0.83	1.42	2.52	3.19	4.16	7.03
8000	0.9616	0.9623	0.9623	0.9622	0.9621	0.9239	8.80	16.94	28.06	40.48	53.61	437.02
16000	0.9617	0.9623	0.9622	0.9621	0.9621	0.8083	19.16	103.13	131.94	212.63	495.86	1494.45
20000	0.9613	0.9621	0.9622	0.9621	0.9620	0.9615	25.16	107.95	220.83	257.94	723.47	2722.22

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time (s).

Parameters: 2000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 0.5$, $k = 6$; 4000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 0.5$, $k = 6$;
8000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 0.8$, $k = 6$; 16000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 0.8$, $k = 6$;
20000: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 0.8$, $k = 6$.

Table 3.7: Pendigits dataset.

Size	**Landmark FN-DBSCAN Rand-Index** r **value**					**FRI**	**Landmark FN-DBSCAN Execution time (s)** r **value**					**FET**
	0.2	0.3	0.4	0.5	0.6		0.2	0.3	0.4	0.5	0.6	
1000	0.9260	0.9220	0.9090	0.9037	0.9018	0.9003	0.06	0.06	0.09	0.17	0.23	0.42
2000	0.9193	0.9151	0.9086	0.9054	0.9030	0.9068	0.05	0.11	0.22	0.38	0.58	1.39
4000	0.9300	0.9185	0.9083	0.9044	0.9025	0.9180	0.11	0.25	0.56	0.98	1.63	5.52
8000	0.9190	0.9154	0.9074	0.9042	0.9020	0.9263	0.55	1.36	2.53	4.09	6.98	66.44
10992	0.9242	0.9160	0.9075	0.9036	0.9021	0.9336	0.45	1.41	3.16	6.45	10.83	120.41

FRI: FN-DBSCAN's Rand-Index. FET: FN-DBSCAN's execution time (s).

Parameters: 1000: $\varepsilon_1 = 0.06$, $\varepsilon_2 = 0.8$, $k = 20$; 2000: $\varepsilon_1 = 0.06$, $\varepsilon_2 = 0.8$, $k = 20$; 4000: $\varepsilon_1 = 0.06$, $\varepsilon_2 = 0.8$, $k = 20$; 8000: $\varepsilon_1 = 0.06$, $\varepsilon_2 = 0.8$, $k = 20$; 10992: $\varepsilon_1 = 0.06$, $\varepsilon_2 = 0.8$, $k = 20$.

cluster, there always existed at least one landmarks in the current cluster such that they were *density-reachable* and *density-connected* wrt. the parameters ε_1, ε_2 and k. So the input orders of data did not influence the representation of data distributions.

To sum up, the input order of data influenced the generated landmarks which were different in positions, neighbors and the membership levels. But the landmarks generated by different input orders provided almost similar representations of the input dataset in data density and data distribution. The result of clustering was not much influenced by the input order of data.

3.3.3 Working with Sparse Data

The capability of dealing with sparse data was tested in the following experiment. The Anchor dataset was prepared as four copies with different values of density, which had strong, normal, medium and low density respectively (See Fig. 3-9(a) to Fig. 3-9(d)).

The data were normalized by Equation (A.3). The parameters were selected for each dataset in order to adapt to different values of density (for the strong density dataset, the parameters were selected as: $\varepsilon_1 = 0.8$, $\varepsilon_2 = 10$, $k = 10$ and $r = 3$; for the normal density dataset, the parameters were selected as: $\varepsilon_1 = 0.6$, $\varepsilon_2 = 10$, $k = 10$ and $r = 3$; for the medium density dataset, the parameters were selected as: $\varepsilon_1 = 0.3$,

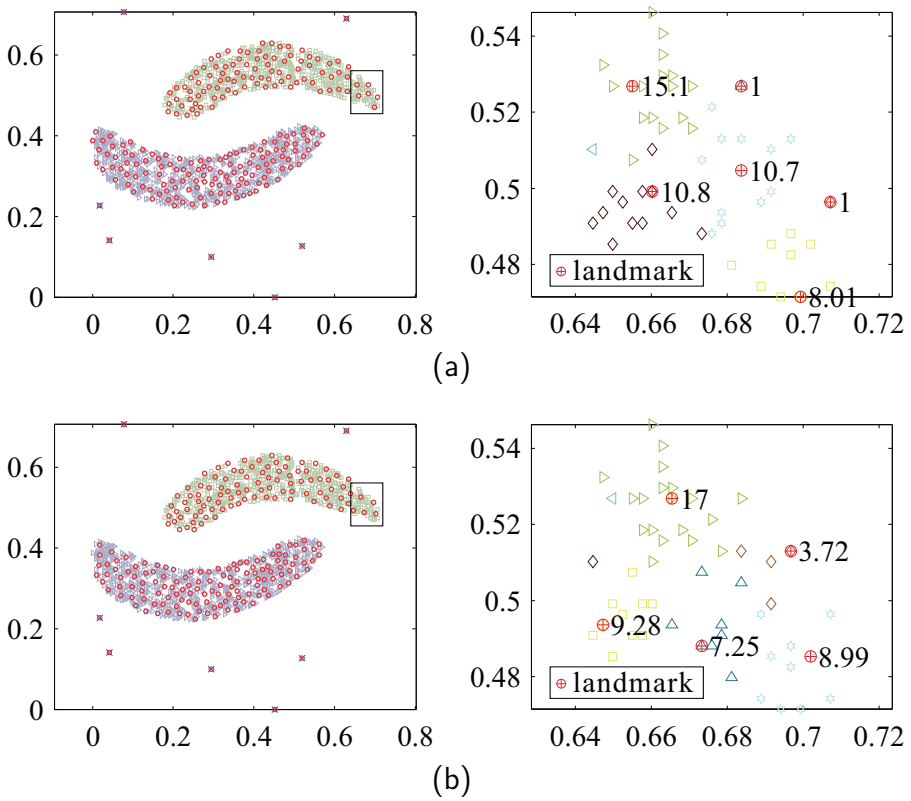


Figure 3-8: Results of generated landmarks with two random input orders. (a) The order 1: the Rand-Index is 1.000 and 231 landmarks are generated. The sum of membership level of all landmarks in the right figure is 46.65. (b) The the order 2: the Rand-Index is 1.000 and 234 landmarks are generated. The sum of membership level of all landmarks in the right figure is 46.23.

$\varepsilon_2 = 5$, $k = 10$ and $r = 3$; for the low density dataset, the parameters were selected as: $\varepsilon_1 = 0.1$, $\varepsilon_2 = 2$, $k = 10$ and $r = 3$). The clustering results are shown from Fig. 3-9(e) to Fig. 3-9(h), respectively. The Rand-Index was used to measure the clustering quality. We see all the results obtained a high score of Rand-Index, which means the algorithm could provide a good clustering quality on these datasets. The authors believe the proposed method is robust in dealing with the low dimensional sparse data. However, working with the high dimensional sparse data is still worth further investigation.

3.3.4 The General Way of Tuning Parameters

In this section, we summarize a general way to select appropriate values of parameters. The landmark FN-DBSCAN has four parameters, ε_1, ε_2, k and r. Among these parameters, ε_1, ε_2 and k are also used in the original FN-DBSCAN algorithm, and the way suggested in the paper of FN-DBSCAN still works, i.e., choosing greater values of parameters ε_1 and k if datasets have high density and choosing smaller values for these two parameters if datasets have low density. The parameter ε_2 is chosen related to the number of noisy data. Greater value is suitable if the number of noisy data is large, but it still requires the expertise and experience of users. The parameter r in the landmark FN-DBSCAN algorithm should be a greater value if the cluster is represented in complex shape, and a smaller value is suitable if the cluster is represented in convex-like shape. The algorithm with a smaller value of r will generate more landmarks, which costs more time to calculate the result, but generally speaking, it can get a better result. In addition, there is a special combination of ε_1 and r suitable for the following case. If a distribution of each cluster is presented in convex-like shape, it is possible that one landmark represents one cluster. In this case, the parameter r can be set as a very small value ($r = 0.2 - 0.9$) and the parameter ε_1 is also set as a small value ($\varepsilon_1 = 0.0001 - 0.05$). With these small values of ε_1 and r, the number of landmarks is expected to equal the number of clusters in the dataset.

3.4 Summary

In this chapter, we proposed a novel clustering algorithm called landmark fuzzy neighborhood DBSCAN (landmark FN-DBSCAN). The presented concept, landmark, was used to represent a subset of the input dataset which made the algorithm efficient on large scale datasets. We gave a theoretical analysis on the time and space complexities of the algorithm, which showed both of them were linear to the size of the dataset. The experiments presented in this chapter also showed landmark FN-DBSCAN was much faster than FN-DBSCAN, almost keeping a good quality of clustering.

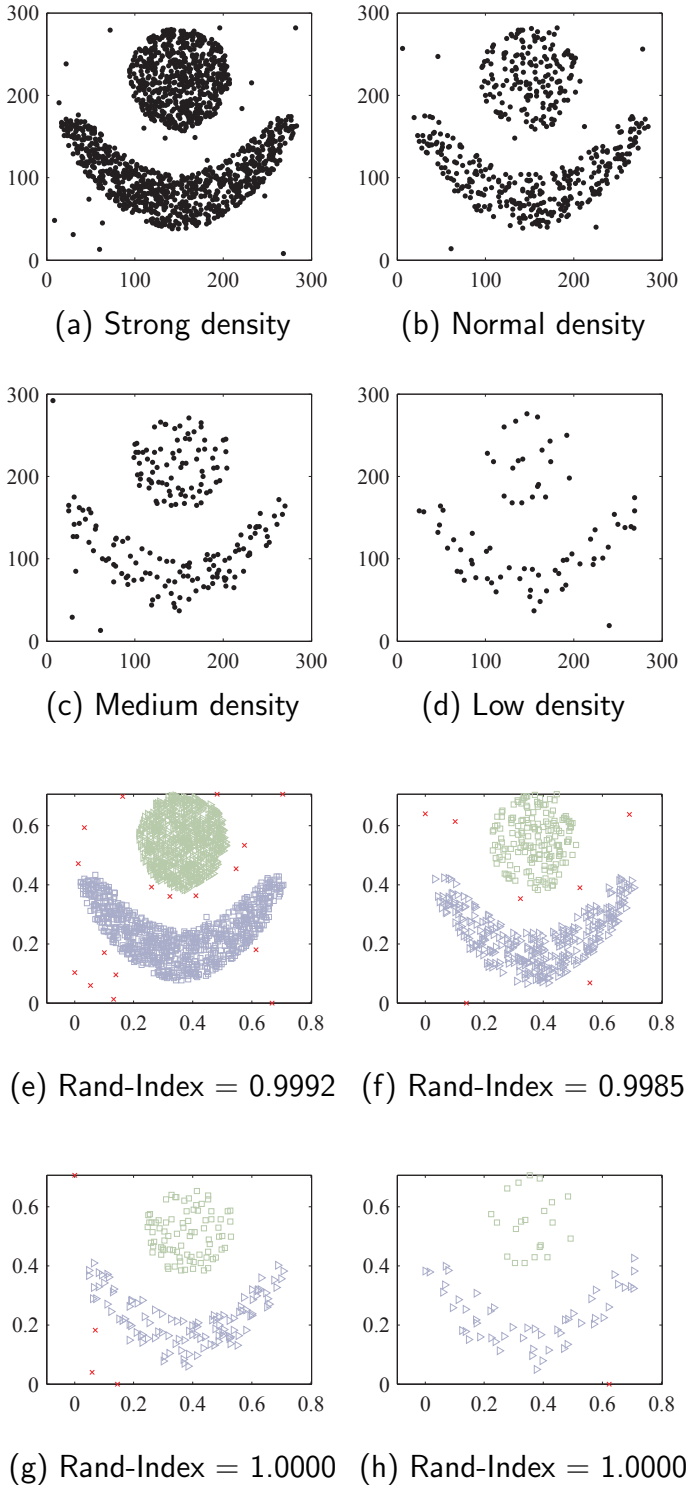


Figure 3-9: The clustering results of the Anchor dataset with different values of density.

Chapter 4

The Locally Adaptive Incremental LBG (LAI-LBG) Algorithm

In this chapter, we first present the motivations of the proposed algorithm, including the problem of using the current landmark generation method (Algorithm 6), as well as the limitations of using both traditional and recently proposed vector quantization methods for the landmark generation task. Then, we propose a new landmark generation method called Locally Adaptive Incremental LBG (LAI-LBG) and present the experimental results.

4.1 The Motivations of the LAI-LBG Algorithm

4.1.1 The Problems of Landmark Generation in the Landmark FN-DBSCAN Algorithm

The landmark generation method, i.e., Algorithm 6, has one critical limitation. The input data distribution represented by the generated landmarks is significantly influenced by the number of landmarks and the input orders. Here, we take a simple experiment to test the performance of Algorithm 6 on generating different numbers of landmarks with different input orders. We carefully selected the parameters in order to let the algorithm generate 250 and 25 landmarks, respectively. In addition, we

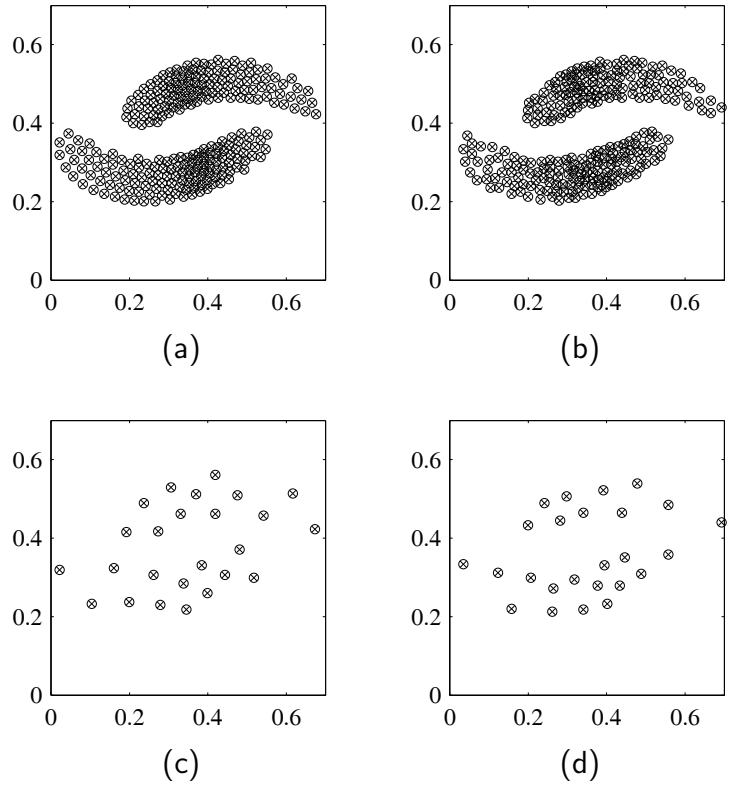


Figure 4-1: The generated landmark positions with different landmark numbers and input orders. 250 landmarks generated with an extreme order (a) and a random order (b). 25 landmarks generated with an extreme (c) and a random order (d).

used an extreme order (data points were input from left to right and from bottom to top) and a completely random order. The experimental object was Banana dataset with 16,437 two-dimensional data points represented in two banana-shaped clusters.

The results are shown in Figure 4-1. We noticed that when the number of landmarks was large, i.e 250, the two clusters could be well presented by the generated landmarks (Figure 4-1(a) and Figure 4-1(b)), which means that the modified FN-DBSCAN algorithm in the later step could easily detect these two clusters based on the generated landmarks. However, when the number was small, i.e. 25, we observed that the generated landmarks were mixed together (Figure 4-1(c)) when the extreme order was used. In this case, the modified FN-DBSCAN algorithm could not easily detect the clusters based on these landmarks. The random order obtained a better result than the extreme order, but the result was still not good enough (Figure 4-1(d)).

Although we could always let Algorithm 6 generate a large number of landmarks to achieve a better representation for the input dataset, it usually means a higher computational cost.

So, one of our aims is to design a new landmark generation method which can provide a good representation of the input dataset with a very small number of landmarks.

4.1.2 Landmark Generation by Vector Quantization Techniques

Since the proposed Algorithm 6 compresses and encodes the input dataset by a set of landmarks, we realize that vector quantization (VQ) techniques can be suitably applied as an "engine" for the landmark generation. VQ is a powerful tool for approximating a probability density function with a finite number of prototype vectors. In addition, it can be used as an encoder for the given dataset and it is not sensitive to the input orders. Let $X = \{x_1, x_2, ..., x_n\}$ be a set of data points and $C = \{c_1, ..., c_g\}$ be a set of prototype vectors. Here, the set C is usually called the *codebook* and the element in C is called a *codeword*. The object of VQ is to find a map, $M : X \to C$, by minimizing a specified quantization error function, e.g. the Mean Quantization Error (MQE, Equation (4.1)). A lower MQE value usually means a better quantization quality.

$$\text{MQE} = \frac{1}{n} \sum_{i=1}^{n} d\left(x_i, \hat{c}\right)^2 \tag{4.1}$$

where $\hat{c}$ is the nearest codeword to x_i determined by Equation (4.2).

$$\hat{c} = \arg\min_{c_j \in C} d\left(c_j, x_i\right) \tag{4.2}$$

Let S be an input space and assume the number of codewords is g. Then the codewords adapted by VQ can divide S into g sub-spaces, which exactly construct a *Voronoi tessellation*. The codewords can be considered as the sites in Voronoi tessellation and each sub-space of S is a *Voronoi region* corresponding to a site.

Here, we denote a Voronoi region with a site c as $R(c)$. Then, the Local Quantization Error (LQE), denoted by E, representing the distortion of a Voronoi region, can be calculated by Equation (4.3).

$$E(c) = \sum_{x \in R(c) \cap X} d\,(x, c)^2 \tag{4.3}$$

Algorithm 9 The LBG algorithm

Input: X, C, β
Output: C

```
E_0 ← 0, E ← 0;
while true do
    E_0 ← E;
    Label each x ∈ X by the nearest codeword according to Equation (4.2);
    Update each codeword c ∈ C by: c = 1/|R(c)| Σ_{x∈R(c)} x;
    E ← Calculate the current MQE according to Equation (4.1);
    if (E_0 − E)/E < β then
        break;
    end if
end while
```

Linde, Buzo, and Gray extended the classical Lloyd's algorithm [100] to vector quantization design, which is often referred to as the "LBG" algorithm [101]. Given a dataset X, a codebook, C and a positive real number, β, as the predefined quantization error threshold, we summarize LBG in Algorithm 9. Figure 4-2(a) shows a quantization result for Banana dataset with 25 vectors. As expected, the vector positions adapted by LBG are much more precise than that calculated by Algorithm 6 (Figure 4-1(c) and Figure 4-1(d)). However, there are still two main limitations, if we directly use LBG as an alternative of Algorithm 6.

Limitation 1: The LBG algorithm cannot always guarantee a good quantization quality, because the quantization procedure significantly depends on the codebook initialization. In general, the codebook is initialized by randomly selecting data points in the given dataset. In this case, different initializations may lead to different quantization results. For example, Figure 4-2(b) shows the quantization result with another different random initialization. Although the quantization result is still better than

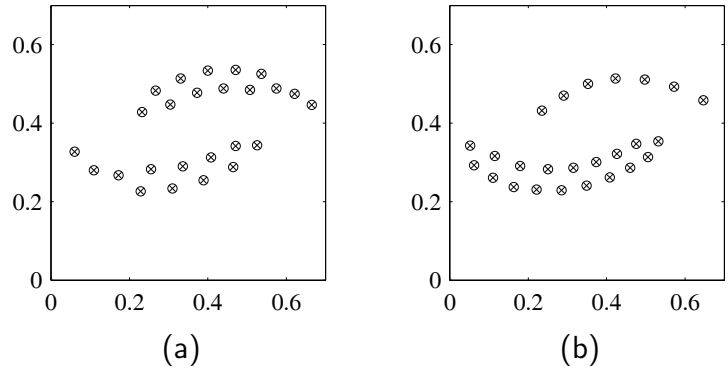


Figure 4-2: The LBG results with different initializations. (a) A "balanced" quantization result with MQE = 0.00076. (b) A "unbalanced" quantization result with MQE = 0.00080.

that of Algorithm 6 (Figure 4-1(c) and Figure 4-1(d)), we can clearly see that the number of adapted codewords used to represent the upper cluster is much smaller than that used to represent the lower cluster. We say it is an "unbalanced" result where the upper codewords have larger contributions to the quantization result than the lower codewords. Since the two banana-shaped clusters have almost the same scale, it is preferable to use a similar number of codewords to represent both of the two clusters. Furthermore, if we use MQE to measure the quantization quality, the result for the case (a) was MQE value at 8.0×10^{-4} and that for the case (b) at 7.6×10^{-4}, which indicates that the case (a) obtained a better quantization result than the case (b). The main reason for the "unbalanced" result is that if more codewords are initialized with the positions around the lower cluster, it is very difficult to move these codewords towards the upper cluster, and vice versa. In addition, pursuing a "balanced" representation plays an important role in the parameter estimation in our proposed method (See more detailed discussions in Section 5.1.2).

Limitation 2: In LBG, the number of codewords must be predefined. The consequential question is how to predefine it. Actually, it is a difficult task for users to determine such a number, because different datasets may have different distributions. In general, in order to represent the data distribution precisely, a dataset with a complex distribution usually needs more codewords than a dataset with a simple dis-

tribution. Although a larger number of codewords helps to get a better quantization quality, it usually results in a critically lower efficiency. So, the number of codewords should be large enough to obtain a good quantization quality and also should be as small as possible to achieve a good efficiency. Unfortunately, it is difficult for users to choose such a suitable number. Therefore, it is preferable for a VQ algorithm to automatically determine a suitable number of codewords and incrementally generate codewords by the algorithm itself.

4.1.3 Considerations about the Adaptive Incremental LBG Algorithm

Thanks to F. Shen and O. Hasegawa, the adaptive incremental LBG (AI-LBG) algorithm [102] provides a good solution for the above two limitations. In AI-LBG, codewords are incrementally inserted until the termination condition is met. Here, the termination condition can be one of the following two criteria. 1) A predefined maximal codeword number. 2) A predefined minimal quantization error. Obviously, the second criterion does not require predefining the number of codewords, which indicates **Limitation 2** can be solved. In addition, a new codeword is always inserted near the codeword with the maximum LQE, which facilitates to achieve a "balanced" quantization result, indicating that **Limitation 1** can be solved. Given a dataset X and a codebook C, the AI-LBG algorithm using predefined minimal quantization error threshold, ξ, is summarized as follows:

1. Initialize the codebook C by randomly selecting a data point from X as the first codeword.

2. Execute the LBG algorithm (Algorithm 9) with full dataset X and codebook C .

3. Calculate the current MQE and record the LQE for each codeword.

4. If the current MQE is greater than the predefined quantization error threshold, ξ, insert a new codeword by the following steps.

- Find the codeword with the maximal LQE:
  $c^\star \leftarrow \arg\max_{c_i \in C} E(c_i)$;
- Randomly select a data point in the Voronoi region with the codeword $c^\star$ as the new codeword, i.e.
  $c \leftarrow x \in R(c^\star)$.
- Add the new codeword to the codebook C, i.e.
  $C = C \cup \{c\}$.

5. Repeat Step 2) to Step 4) until MQE is no greater than the threshold ξ.

6. Remove the codewords whose LQE is zero or the smallest one, then go to Step 2) to insert new codewords. The algorithm will terminate when such a removal-insertion process cannot decrease the number of codewords.

However, we notice that the procedure, from Step 2) to Step 4), is a loop where each iteration needs an execution of the LBG algorithm on the whole dataset and the full codebook, which indicates a high computational cost. We say the above method is the AI-LBG with global adaption, i.e. the *AI-LBG* (*global*) algorithm. F. Shen and O. Hasegawa also introduced a local adaption method to reduce the computational cost. In each iteration, the LBG algorithm is executed with only two codewords, i.e $\{c^\star, c\}$, and the data points in $R(c^\star)$. We say this method is the AI-LBG with local adaption, the *AI-LBG* (*local*) algorithm. Although the AI-LBG (local) has improved the efficiency of AI-LBG (global), only considering the local adaption cannot guarantee the quantization result as good as that of AI-LBG (global). In addition, in order to achieve a high efficiency by local adaption, the Voronoi regions cannot be globally calculated in each iteration. In this case, the data points cited by the codewords may not be exactly the same as the corresponding Voronoi regions, which may cause an unacceptable distortion if we use it as an encoder for the input dataset.

4.1.4 The Requirements of a Vector Quantization Method for Landmark Generation

Based on the above discussions, a desired vector quantization method for landmark generation should have the following advantages:

1. The input orders will NOT influence the quantization result.
2. The algorithm can always achieve a "balanced" result with a high quantization quality which is not dependent on the initialization.
3. The number of landmarks is NOT a prerequisite.
4. The high efficiency is a vital feature.

4.2 Locally Adaptive Incremental LBG

In this section, we first introduce several basic concepts in Section 4.2.1, then we present a new incremental vector quantization method for landmark generation called Locally Adaptive Incremental LBG (LAI-LBG), which can provide high quantization quality similar to that of AI-LBG (global) and significantly improve the efficiency. Based on LAI-LBG, we further propose a novel clustering algorithm as an improvement of the landmark FN-DBSCAN algorithm in Chapter 5.

4.2.1 Basic Concepts

Since we want to use an incremental VQ technique to generate landmarks and encode the given dataset by the generated landmarks, the related definitions are modified as follows:

Definition 10 (landmark) *Given a dataset X as an n-by-m matrix, where n is the number of data points and m is the dimensionality of data points, a landmark, denoted as l, is a triplet:*

$$l = \langle \mathbf{V}, C, N_{\varepsilon_1} \rangle \tag{4.4}$$

where:

- $\mathbf{V}$ *is an* m*-dimensional vector in the space* R^m*, called the vector of the landmark* l*.*
- C *is a subset of* X*, called the citations of the landmark* l *and we say that the data point* x *is cited by* l *if* $x \in C$*.*
- N_{ε_1} *is a set of landmarks called the* $fuzzy$*-neighborhood of* l *which will shortly be defined by Definition 11.*

In the following discussions, we denote the three elements of a landmark l by $V(l)$, $C(l)$ and $N_{\varepsilon_1}(l)$, respectively.

Comparing to Definition 8, we introduced a new concept, the citation C, to a landmark in order to describe a subset of the input dataset. The citation is simply defined as an abstract set, which means the manipulation of cited data points only depends on the algorithms that adopt the landmark concepts. The fuzzy-neighborhood, N_{ε_1}, has some specialized criteria by Definition 11. The concept, citations, is only used for the relationship between data points and landmarks, while the fuzzy-neighborhood is only used for the relationship between landmarks.

Definition 11 (fuzzy-neighborhood of a landmark) *Given a landmark set* $Ł$ *and a landmark* l $(l \in Ł)$*, the fuzzy-neighborhood of* l *is a subset of* $Ł$ *as:*

$$N_{\varepsilon_1}(l) = \{l' \in Ł \mid f_l(l') \geq \varepsilon_1\} \tag{4.5}$$

where the function $f_l(l')$ *is given by Equation (4.6) and the parameter* ε_1 *is a positive real number. We say* l' *is a neighbor of* l *if* $l' \in N_{\varepsilon_1}(l)$*.*

$$f_l(l') = \exp\left(-\left(k \cdot \frac{d(\mathbf{V}(\mathbf{l}), \mathbf{V}(\mathbf{l}'))}{d^{max}}\right)^2\right) \tag{4.6}$$

where k *is a positive real number for controlling the neighborhood radius and* d^{max} *is the maximum distance between* l *and all the other landmarks in* $Ł$*.*

On the basis of Definition 10 and Definition 11, the *landmark-cardinality* can be calculated by Definition 12.

Definition 12 (*landmark-cardinality*) *Let l be a landmark, then the landmark-cardinality of l can be calculated by:*

$$card(l) = \sum_{l_i \in N_{\varepsilon_1}(l)} f_l(l_i) \cdot |C(l_i)| \tag{4.7}$$

where $N_{\varepsilon_1}(l)$ is the fuzzy-neighborhood of l and $C(l_i)$ is the citation of a landmark l_i which is a neighbor of l. Given a positive real number, ε_2, a landmark is called a core-landmark if its cardinality is no less than ε_2, i.e. $card(l) \geq \varepsilon_2$.

According to Definition 10, a landmark is assigned with a vector, which indicates that a landmark can also be considered as a codeword. So, in the following discussions, codewords and landmarks are collectively referred to as "landmarks".

4.2.2 The Proposed Method

The main reason for the low efficiency of AI-LBG (global) is the heavy calculation with the whole dataset and the full landmark set in each iteration. Although the algorithm needs to find the right positions for the newly inserted landmark and the corresponding nearby landmarks, we notice that it is not necessary to use the whole dataset and all landmarks to finish this adaption. Here, we show a simple example for this issue.

Figure 4-3 shows a snapshot of an adaption for three clusters with several landmarks. The landmark l_1 has the maximal LQE value (Figure 4-3(a)), and a new landmark should be accordingly inserted near l_1 in order to achieve a "balanced" quantization result. Actually, an approximated adaption can be finished just by the data points only in Cluster 3 and the two landmarks, l_1 and the newly inserted landmark (Figure 4-3(b)), which means only a small part of data points and a few corresponding landmarks are involved in the current iteration so that the computational cost can be significantly reduced. We say that such an adaption is a local adaption.

This idea performs like what AI-LBG (local) does. However, the main drawback of this technique is the difficulty in obtaining a high quantization result, simply because the scale of data points that it considers is too small to represent a local area.

In order to efficiently obtain a high quantization quality, our solution is to divide the conventional VQ procedure into two parts: vector initialization and vector quantization. The initialization part is greatly inspired by the AI-LBG algorithm, i.e. monitor the LQE value for each landmark and always insert a new landmark near the one with the highest LQE value so that the algorithm can provide a balanced landmark generation result. However, we develop a new local adaption method to improve the efficiency. Then, the vector quantization part in the later step will execute a modified LBG algorithm on the whole dataset together with the full generated landmarks so that a global quantization result can be obtained. Here we must note that the later step can also achieve a very good efficiency because only a few number of iterations is required, which benefits from a well-balanced landmark positions by the former initialization part.

Now, we present the new local adaption in detail.

First, we update the quantization error measures, i.e LQE and MQE. In Equation (4.3), the LQE value is obtained according to the Voronoi regions of landmarks. However, such a calculation will require a high computational cost, if we do it for each iteration. In order to improve the efficiency, we calculate an approximated LQE

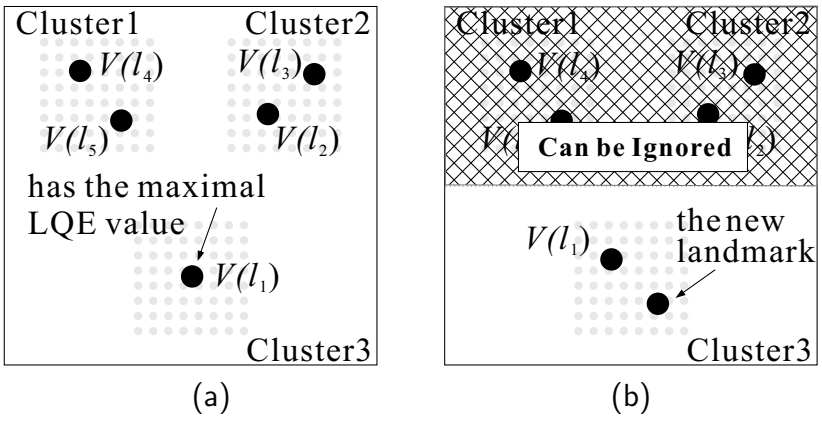
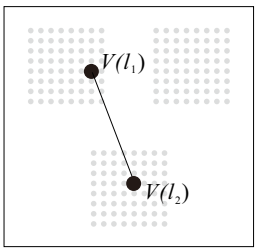
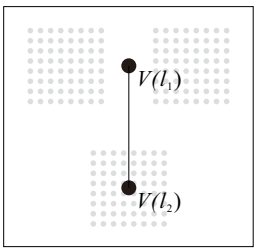


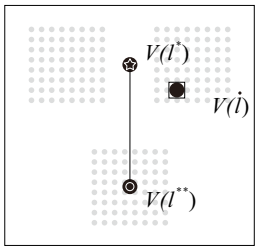
Figure 4-3: An illustration of the LBG algorithm with a Locally Adaptive Incremental Initialization.



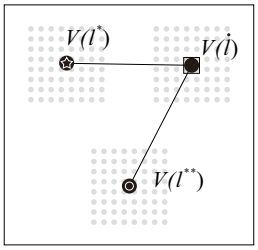
(a) Initialize two landmarks by randomly selecting two data.



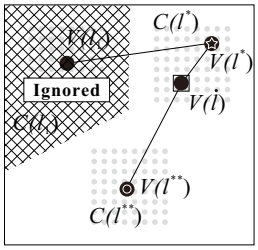
(b) Execute the modified LBG algorithm.



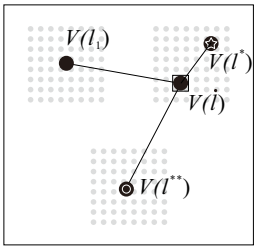
(c) If the current MQE > ξ , then, insert a landmark.



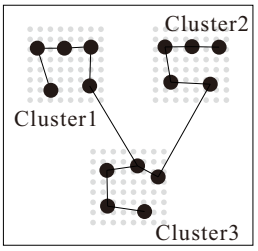
(d) Execute the modified LBG algorithm. Update the connections.



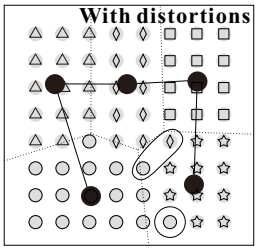
(e) Insert another landmark and execute the modified LBG only with three landmarks (l^*, l^{**} and $\dot{l}$) and the data belonging to l^* or l^{**}.



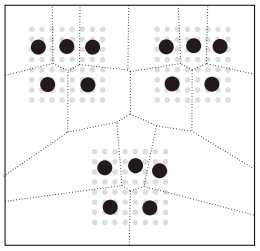
(f) Connections are updated to maintain locally nearby relationships.



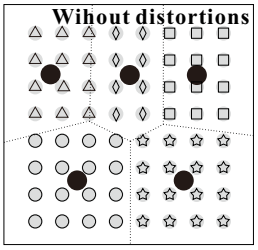
(g) The initialization part ends when MQE ⩽ ξ .All clusters are presented by the generated landmarks.



(h) The distortion between data citations and the real Voronoi regions in Cluster1.



(i) Execute the modified LBG algorithm on the whole data set with all landmarks to form the final Voronoi Tessellation.



(j)The final Voronoi regions in Cluster 1. The distortions are eliminated.

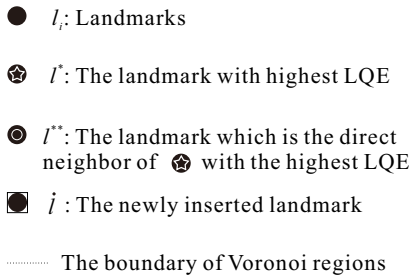


Figure 4-4: Illustrations for the Locally Adaptive Incremental LBG Algorithm.

value from the citations of landmarks. Let l and X be a landmark and the given dataset, respectively. The LQE of l can be approximately calculated by Equation (4.8).

$$E(l) = \sum_{x \in C(l) \cap X} d\left(x, V(l)\right)^2 \tag{4.8}$$

Then, let Ł be a set of landmarks. The MQE value can be obtained by Equation (4.9).

$$\text{MQE} = \frac{1}{n} \sum_{l_i \in \text{Ł}} E(l_i) \tag{4.9}$$

where n is the number of data points in X.

Second, comparing to the AI-LBG (local), we use a bigger local area in order to obtain a better quantization quality. We use two special landmarks, i.e. $l^\star$ and $l^{\star\star}$, to represent a local area. Here, $l^\star$ is the landmark which has the maximal LQE value among all the landmarks indicating a small area with the highest quantization error. The landmark $l^{\star\star}$ is close to $l^\star$ which is pointing out a nearby small area which also has a high quantization error. Thus, these two small areas construct a local area in the input space and we know that this local area potentially has a higher quantization error than the other areas. The newly inserted landmark, $\dot{l}$, is always selected from the citation of $l^\star$, $C(l^\star)$, which guarantees that the quantization error will be reduced for this local area. In this procedure, only the data points belonging to $C(l^\star)$ or $C(l^{\star\star})$ and three landmarks $\{l^\star, l^{\star\star}, \dot{l}\}$ participate in the adaption.

Third, we develop a simple and effective accelerating method for the search of $l^{\star\star}$. We organize the generated landmarks in a graph where nodes represent the landmarks and connections (edges) represent the so-called *nearby-relationship* between two landmarks. Here, the *nearby-relationship* indicates that the two landmarks are very close to each other in the input space. For any pair of landmarks in the graph, we say that they are *directly-adjacent* to each other or they are the *directly-adjacent neighbors* to each other if there is a connection between them. The graph is dynamically maintained during the whole initialization part. At the beginning, the graph is initialized by two directly-adjacent landmarks (Figure 4-4 (a)). After inserting a

landmark, $\dot{l}$, the old connection between $l^{\star}$ and $l^{\star\star}$ is removed. Then, create two connections representing the two shortest edges in the triangle with $\{l^{\star}, l^{\star\star}, \dot{l}\}$ as the vertices (Figure 4-4 (d)). In addition, if the landmark $l^{\star}$ has more than one directly-adjacent neighbor before the insertion, then the connections will be updated in order to maintain the locally nearby-relationship. For example, in Figure 4-4 (e), the landmark $l^{\star}$ has two directly-adjacent neighbors, i.e. l_1 and $l^{\star\star}$. If the newly inserted landmark, $\dot{l}$, is closer to l_1 than $l^{\star}$, then the connection, $c(l^{\star}, l_1)$, will be removed and a new connection, $c(\dot{l}, l_1)$, will be created (Figure 4-4 (f)).

Now, we summarize the LAI-LBG algorithm as follows:

1. **Initialization part.**

   (a) The algorithm starts with two landmarks $\{l_1, l_2\}$ whose vectors are initialized by randomly selecting two data points in the dataset X. Then, create a connection $c(l_1, l_2)$ between l_1 and l_2 and add l_1 and l_2 into the landmark set Ł (Figure 4-4 (a)). Next, execute the algorithm 10 which is a modified LBG algorithm (Figure 4-4 (b)). Record the LQE values for l_1 and l_2.

   (b) Repeat the following sub-steps until MQE $\leq \xi$.

      i. Find the landmark $l^{\star}$ with the highest LQE ($l^{\star} \in$ Ł).

$$l^{\star} = \arg\max_{l_i \in Ł} E(l_i) \tag{4.10}$$

      ii. Let $\hat{N}$ be the set of directly-adjacent-neighbors of $l^{\star}$. Search $\hat{N}$ for a landmark which has the highest LQE value.

$$l^{\star\star} = \arg\max_{l_i \in \hat{N}} E(l_i) \tag{4.11}$$

      iii. Create a new landmark, $\dot{l}$, by randomly selecting a data point in $C(l^{\star})$. Add $\dot{l}$ to Ł (Figure 4-4 (c)).

      iv. Execute Algorithm 10 only with the data points which belong to $C(l^{\star}) \cup C(l^{\star\star})$ and only with three landmarks, i.e. $l^{\star}$, $l^{\star\star}$ and $\dot{l}$ (Figure

Algorithm 10 The modified LBG algorithm

Input: Ł, X, β
Output: Ł

$E_0 \leftarrow 0, E \leftarrow 0$;
while true **do**
  $E_0 \leftarrow E$;
  Empty the citations of each landmark ($l_i \in$ Ł);
  For each data point $x \in X$, do $C(l_i) \leftarrow C(l_i) \cup \{x\}$, if there is no landmark l_j, ($l_j \in$ Ł), such that $d(V(l_j), x) < d(V(l_i), x)$;
  For each landmark $l_i \in$ Ł, update the landmark vectors by $V(l_i) = \frac{1}{|C(l_i)|} \sum_{x \in C(l_i)} x$;
  $E \leftarrow \sum_{l_i \in Ł} \sum_{x_j \in C(l_i)} d(V(l_i), x_j)^2$;
  if $(E_0 - E)/E \leq \beta$ **then**
    break;
  end if
end while

4-4 (d)).

v. Remove the connection, $c(l^\star, l^{\star\star})$, then create two connections representing the two shortest edges in the triangle with $\{l^\star, l^{\star\star}, \dot{l}\}$ as the vertices (Figure 4-4 (d)).

vi. If a landmark l_i ($l_i \in \hat{N}/\{l^{\star\star}\}$) satisfies that $d(V(l_i), V(\dot{l})) < d(V(l_i), V(l^\star))$, then remove the connection $c(l_i, l^\star)$ and create a connection $c(l_i, \dot{l})$ (Figure 4-4 (e) and (f)).

(c) Remove the landmarks whose LQE value is zero, i.e. those landmarks with empty citations are removed.

2. **Vector quantization part.**

(a) Execute the Algorithm 10 with the whole dataset X and full landmark set Ł.

(b) If the algorithm terminated, output the landmark set Ł.

Based on the previous discussions, the initialization part only takes the local adaption. If we compare the citations of landmarks with the data points belonging to the corresponding Voronoi regions, sometimes, distortions may exist. We show an

example distortion for Cluster 1 (the left-upper cluster in Figure 4-4 (g)) in Figure 4-4 (h) where the data points cited by different landmarks are marked by different shaped markers. In order to eliminate such distortions, in the vector quantization part, Algorithm 10 is executed again with the whole dataset and the full landmark set. After that, for each landmark, the citations are exactly the same as the data points belonging to the Voronoi region (Figure 4-4 (j)). Please note that the landmarks generated by the initialization part are almost already in the final positions in the input space (Figure 4-4 (g)), which means that very few iterations are required in this procedure.

4.3 Experimental Results

In this section, we will evaluate the performance of the LAI-LBG algorithm for vector quantization. All the experiments were conducted by Intel Core i7 3.40 GHz CPU with 8GB RAM running on Windows 7 64-bit operating system. All the algorithms involved were implemented in Matlab language and were executed on the platform of Matlab 2013a 64-bit version. The datasets were normalized by Equation (A.3).

In this section, we focus on the performance evaluation of the LAI-LBG algorithm for vector quantization. In the experiments, we first used Cantor distribution [103] to test whether or not the algorithm could achieve a balanced quantization result. Then, we further used the Banana dataset to evaluate the performance of landmark generation. We tested the performance with different number of landmarks. All the experiments were designed with the consideration about both quality and efficiency. All the results shown in this section were the average value of 10 times runs and compared with LBG, AI-LBG (local) and AI-LBG (global).

In the first experiment, we used a three-level Cantor distribution which consisted of 4096 data points symmetrically distributing in 16 small clusters. In order to obtain a fair comparison for all the tested algorithms, we set the minimized quantization threshold β by zero in order to achieve the best convergence for all algorithms. In addition, we directly set the number of landmarks by 16 for the classic LBG algorithm

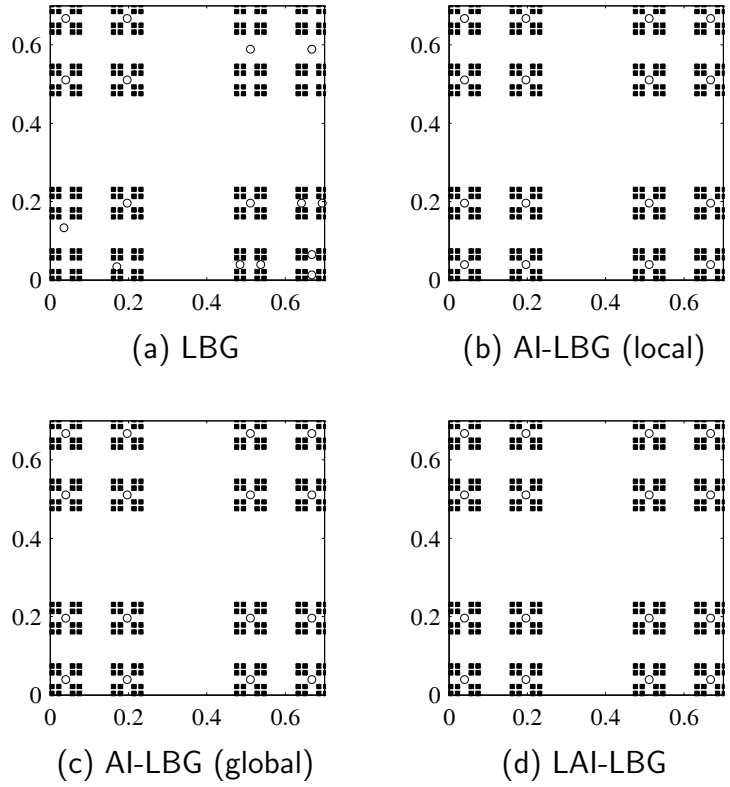


Figure 4-5: The results for the Cantor dataset.

and carefully selected appropriate values for parameter ξ for the rest of algorithms so that all the algorithms could generate 16 landmarks. The landmarks calculated by LBG, AI-LBG (local), AI-LBG (global) and LAI-LBG are shown in Figure 4-5(a), 4-5(b), 4-5(c) and 4-5(d), respectively. We can see that the classic LBG algorithm mis-placed several landmarks: less landmarks for the right-upper and left-bottom clusters but more landmarks for the right-bottom clusters, which was an unbalanced quantization result for the symmetrical Cantor dataset. On the other hand, AI-LBG (local), AI-LBG (global) and the proposed LAI-LBG could achieve a perfectly balanced distribution for landmarks. Furthermore, we recorded the MSE values in Table 4.1, where the AI-LBG (global), AI-LBG (local) and LAI-LBG had much smaller MSE values than LBG, which further proved that they achieved a higher quantization quality than LBG. In addition, we also recorded the execution time in Table 4.1, where we noticed a significant difference on efficiency. Although the classical LBG algorithm obtained the highest efficiency, its quantization quality was too poor.

The proposed LAI-LBG achieved about 34% faster than AI-LBG (global) and the difference became more significant when the number of landmarks became larger (see the following experiment). Although the AI-LBG (local) algorithm was slightly faster than the proposed method, the distortion existed in its result, which means the data points cited by landmarks might not be the data points in the real Voronoi regions. We show the distortion comparisons in the following experiment.

Table 4.1: The efficiency and quality comparison for the Cantor dataset.

	LBG	**AI-LBG (local)**	**AI-LBG (global)**	**LAI-LBG**
MSE	0.0037	0.0015	0.0015	0.0015
Time	0.0102	0.0165	0.0547	0.0185

Next, we used the Banana dataset to further examine the performance of the proposed method. This time, the algorithms were tested with different number of landmarks, i.e. 25, 50, 100, 200, 400 and 800. The results are shown in Table 4.2.

Table 4.2: The Results for the Banana dataset.

N[1]	LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	MSE	Time	MSE	Time	MSE	Time	MSE	Time
25	7.3095×10^{-4}	0.4339	7.7337×10^{-4}	0.0629	7.0941×10^{-4}	2.6194	7.1093×10^{-4}	0.3875
50	3.4792×10^{-4}	0.9000	3.6344×10^{-4}	0.1308	3.3963×10^{-4}	10.5698	3.4259×10^{-4}	0.7567
100	1.7153×10^{-4}	1.5017	1.8678×10^{-4}	0.3370	1.6685×10^{-4}	29.8033	1.6858×10^{-4}	1.3818
200	8.5703×10^{-5}	2.1506	8.9822×10^{-5}	1.0636	8.3077×10^{-5}	83.6323	8.3986×10^{-5}	1.6649
400	4.3368×10^{-5}	2.7906	4.6129×10^{-5}	3.7877	4.1721×10^{-5}	221.5666	4.2055×10^{-5}	2.5092
800	2.2255×10^{-5}	3.5422	2.2377×10^{-5}	14.1099	2.0957×10^{-5}	566.1216	2.1119×10^{-5}	3.7987

[1]N indicates the number of landmarks.

We noticed that for AI-LBG (global) and the proposed method, both of them obtained a similar high quantization quality, but the proposed method saved over 99% of time than AI-LBG (global). The AI-LBG (local) achieved a high efficiency when the number of landmarks was small, but the quantization quality was poor comparing to that of AI-LBG (global) and the proposed method. The main reason is that a significant distortion between the landmark citations and the real Voronoi regions could be observed in each iteration and the final result, which means that only the local adaption cannot make sure a consistency between the landmark citations and the real Voronoi regions. On the other hand, there were no such distortions in

the LAI-LBG results. Here, we show an example of the distortions in Figure 4-6 and we used Rand-Index (Appendix A.3.) to measure how much the distortions were.

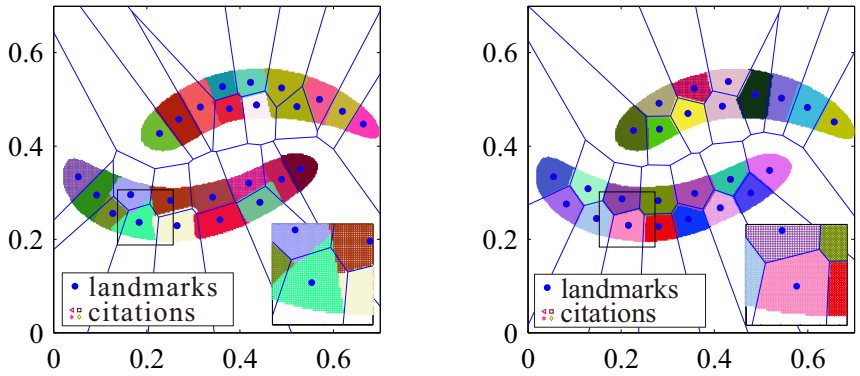


Figure 4-6: The distortion comparisons between the landmark citations and the real Voronoi regions when the number of landmarks were 25. The citations of different landmarks are marked by different colors.

Table 4.3: The distortions comparison in the results of AI-LBG (local) and LAI-LBG measured by Rand-Index.

The number of landmarks	**Rand-Index** AI-LBG (local)	LAI-LBG
25	0.9907	1.0000
50	0.9960	1.0000
100	0.9971	1.0000
200	0.9988	1.0000
400	0.9992	1.0000
800	0.9997	1.0000

We summarized the results in Table 4.3. For the AI-LBG (local) algorithm, the distortions existed throughout the experiment, whereas there was no distortion at all in the LAI-LBG result. For AI-LBG (local), the distortion reduced with the increase of the landmark numbers. When the number of landmarks was small, e.g. 25, the distortion resulted in a Rand-Index value at 0.9907 (Figure 4-6). When the number of landmark increased, the size of the landmark citations shrunk, which made the data citations approximately equal to the corresponding Voronoi regions. This reduced the distortions, e.g. the Rand-Index value was approximately equal to 1 when the

number of landmarks was 800, which indicated that almost no distortion existed. However, according to our previous experiment result, Table 4.2, the efficiency of AI-LBG (local) deteriorated when the number of landmarks was large, because AI-LBG (local) spent more time on the insertion-removal step when the number of landmarks increased.

4.4 Summary

In this chapter, we proposed a novel vector quantization method called Locally Adaptive Incremental LBG (LAI-LBG) which serves as the landmark generation method. Compared to the other methods, such as the classical LBG algorithm and the most recently proposed methods including the landmark generation method used in the landmark FN-DBSCAN algorithm, AI-LBG (global) and AI-LBG (local), the proposed LAI-LBG algorithm has the following new features:

1. The algorithm has a significantly higher efficiency than the other methods which can provide a similar quantization quality.

2. The algorithm is not sensitive to the initializations and the input data orders, which means the generated landmarks can be considered as a balanced representation of the input dataset.

3. The algorithm incrementally inserts and removes landmarks with no requirement of predefining the number of landmarks.

From the Cantor dataset experiment, we conclude that the classic LBG algorithm fails to provide a well-balanced distribution for the landmarks, which is a key requirement of the proposed clustering algorithm. The AI-LBG (global) can obtain excellent quantization results, but the low efficiency is not acceptable when it is used as a part of an efficient clustering algorithm. The AI-LBG (local) has a good efficiency when the number of landmarks is small. Although for some datasets, e.g the datasets with smaller cluster numbers and simpler distributions, the number of landmarks may not be required to be large, the distortions existing in the result of

AI-LBG (local) causes a poor quantization quality, which may finally lead to bad clustering results. The proposed method, LAI-LBG, can achieve a good performance both on quantization quality and efficiency, which indicates that it is suitable to be used in the implementation of an efficient clustering algorithm.

Chapter 5

The Improvement of the landmark FN-DBSCAN Algorithm

On the basis of the LAI-LBG algorithm, in this chapter, we propose an improved landmark FN-DBSCAN algorithm. We also discuss about the parameter selection and outliers detection for the improved algorithm.

5.1 The Improved landmark FN-DBSCAN Algorithm

5.1.1 The Proposed method

Given a dataset, our basic idea is summarized as follows:

1. Use LAI-LBG to efficiently obtain a set of landmarks. Then the input space is divided into several sub-spaces represented by the generated landmarks corresponding to the Voronoi regions.

2. Detect outliers for each Voroni region by Grubbs' test (See A.1 for detecting outliers by Grubbs' test).

3. Use a modified FN-DBSCAN clustering algorithm (detailed in Algorithm 11) to get the clustering information about the generated landmarks.

4. Label the input data points according to the clustering information about landmarks.

Here, we use Banana dataset again to illustrate how the proposed algorithm works. However, at this time, the Banana dataset has 16, 535 data points and also contains some outliers (Figure 5-1(a)).

After Step 1), the input space has been divided into several Voronoi regions with the generated landmarks as their sites (Figure 5-1(b)). We notice that, for each Voronoi region, the local data distribution can approximately be considered as a "convex-shaped" distribution with the landmark vector as the gravity center. Thus,

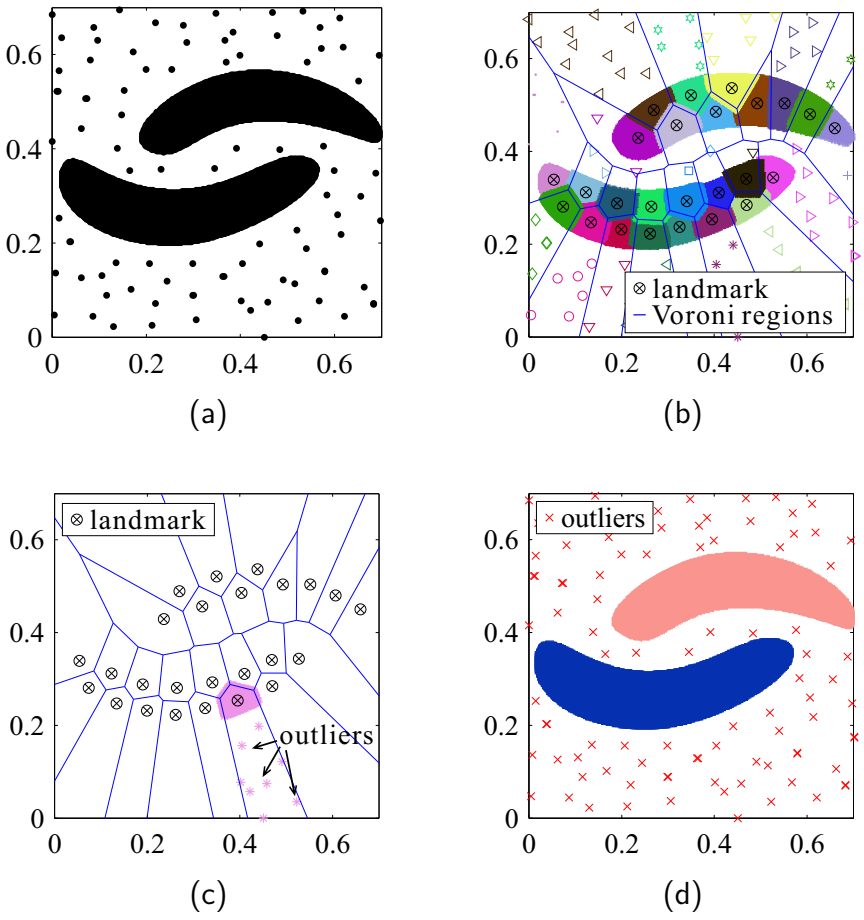


Figure 5-1: An illustrated clustering procedure of the proposed algorithm (a) The input dataset (Banana dataset with outliers). (b) After the Step 1), 25 landmarks are generated and 25 corresponding Voronoi regions are formed. (c) The outliers can be detected and removed for each Voronoi region. (d) The final clustering result.

for each Voronoi region, if we calculate the distances between each one of the data points and the landmark vector, such distance values should approximately be a Gaussian distribution. Therefore, the Grubbs' test is applicable to the outliers detection (Figure 5-1(c)). Fortunately, we do not have to re-calculate such distances because in the former step, the execution of LAI-LBG, we had already finished this calculation when it measured the MQE value. So, just reusing them is an efficient manner to implement the Grubbs' test.

Algorithm 11 A Further Modified FN-DBSCAN Algorithm

Apply Definition 11 and Definition 12 as the fuzzy-neighborhood function and the landmark cardinality, respectively.

Input: Ł, k, ε_1, ε_2
Output: $\mathcal{C}$ and outliers label

Mark all landmarks as "unassigned";
$\mathcal{C} \leftarrow \emptyset$;
for all l in Ł **do**
  if l is marked as "unassigned" **then**
    $N_{\varepsilon_1}(l) \leftarrow$ Find the fuzzy-neighborhood of l wrt. ε_1 and k;
    $card(l) \leftarrow$ Calculate the *cardinality* of l;
    if $card(l) \geq \varepsilon_2$ **then**
      Mark l as "assigned";
      Create a new cluster C, initialized by $C \leftarrow \{l\}$;
      Create a queen S, initialized by $S \leftarrow N_{\varepsilon_1}(l)$;
      for all p in S **do**
        $C \leftarrow C \cup \{p\}$;
        Mark p as "assigned";
        $N_{\varepsilon_1}(p) \leftarrow$ Find fuzzy-neighborhood of p wrt. ε_1 and k;
        $card(p) \leftarrow$ Calculate the *cardinality* of p;
        if $card(p) \geq \varepsilon_2$ **then**
          Add all data points in $N_{\varepsilon_1}(p)$ to the end of the queen: $S \leftarrow S \cup N_{\varepsilon_1}(p)$;
        end if
      end for
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$;
    end if
  end if
end for
Mark all remaining "unassigned" landmarks as outliers;

Next, the modified FN-DBSCAN is used to obtain the clustering information about landmarks, which is detailed in Algorithm 11. Since each data point is cited

by only one landmark, we can easily label the input data points according to the clustering information about landmarks. For example, let x_i and x_j be two data points which are cited by landmarks l_p and l_q, respectively, i.e. $x_i \in C(l_p)$ and $x_j \in C(l_q)$. If l_p and l_q are in the same cluster according to Algorithm 11, then give the same cluster label to x_i and x_j.

5.1.2 Tuning the Parameters

There are six parameters in the proposed method. Most of them have a typical range for the selection or can be easily estimated for the normalized datasets (See Appendix A.1 for the method of data normalization).

The parameter ξ is the threshold for the minimized MQE value in the quantization convergence procedure of the LAI-LBG algorithm. The smaller value of ξ is selected, the higher quantization quality is achieved, but more iterations are required. As with the most VQ techniques, the minimized MQE value can typically be in the range of $[0.0002, 0.001]$. Selecting 0.0002 for the parameter ξ can achieve both high efficiency and high clustering quality in our experiments (see more results in Section 5.2).

The parameter α is the significance level used in the Grubbs' test. In general, such a significance level should be a small value. Using 0.05 can achieve a very good result in most cases.

The parameter β is the convergence threshold for the minimized MSE value in the modified LBG algorithm (Algorithm 10). Such a threshold is also a very small value. Setting β with zero will achieve the best quantization quality. In general, a typical value can be in the range of $[0.001, 0.005]$ in order to achieve a faster convergence.

The parameters k and ε_1 is a parameter pair used to control the radius of the fuzzy-neighborhood of a landmark (Definition 11). Now, we give an estimation method for these two parameters. Figure 5-2(a) illustrates two adjacent Voronoi regions. If landmarks l_p and l_q are in the same cluster, they must be neighbors to each other according to Definition 11. If so, the the distance between $V(l_p)$ and $V(l_q)$ must be appropriately suitable for determining the parameters k and ε_1. Let such a suitable distance be $\dot{d}$, then the parameter k can be obtained by Equation (5.1) which is

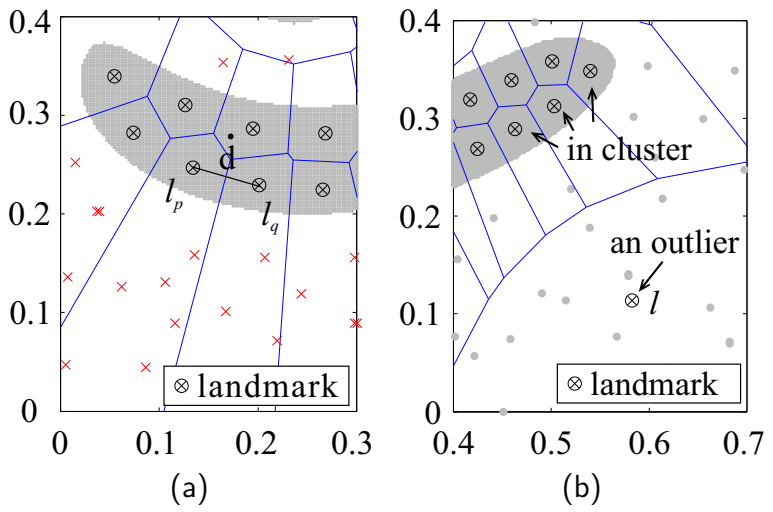


Figure 5-2: (a) Estimating the suitable distance between two neighbor landmarks. (b) The landmark l will be considered as an outlier.

transformed from Equation (4.6).

$$k = (d^{max}/\dot{d}) * \sqrt{-\log(\varepsilon_1)} \tag{5.1}$$

Since determining the parameter k is related to ε_1, we can arbitrarily select a value for ε_1 ($\varepsilon_1 \in (0, 1)$), and then Equation (5.1) will obtain a corresponding value for k. Now, the only problem is how to know the "suitable distance", $\dot{d}$.

Actually, estimating the $\dot{d}$ value can be inspired from how to determine the parameter *Eps* of DBSCAN. [21] proposed an effective method to solve this problem (called M. Ester's method). Their main idea is to calculate, for each data point x

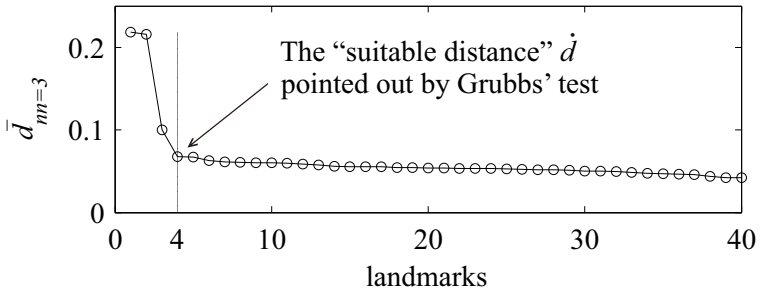


Figure 5-3: The sorted mean distances between landmarks and their corresponding top-three neighbors for Banana dataset with 40 landmarks generated.

in the dataset, the distance from x to its m-th nearest-neighborhood, where, typically, m is set to 4 in a method called 4-*dist*. After that, sort all such distances in descending order and show them as a curve as in Figure 5-3. The suitable value for the parameter *Eps* is determined as the distance associated with the leftmost point from which the curve tends to flatten. However, DBSCAN is very sensitive to the estimated *Eps* value because of its crisp-neighborhood model, which indicates such a point on the curve must be precisely located. Fortunately, FN-DBSCAN reduced the parameter sensitivity because of the soft neighborhood model adopted [24]. Since DBSCAN and FN-DBSCAN share the essential concepts, the M. Ester's method for estimating the parameter *Eps* of DBSCAN is still valuable for the estimation of the parameters k and ε_1 of FN-DBSCAN. Now, we extend the M. Ester's method to estimate the parameter k and ε_1 in low dimensional space as Algorithm 12.

Algorithm 12 Estimate the parameters k and ε_1

Input: Ł

Output: k, ε_1

1: $\bar{D}_{nn=3} \leftarrow$ For each landmark $l \in$ Ł, calculate the mean distance, $\bar{d}_{nn=3}$, between $V(l)$ and its top-three nearest-neighbors. Let $\bar{D}_{nn=3}$ be the set of all $\bar{d}_{nn=3}$;
2: $O \leftarrow$ Use Grubbs' test to detect the significantly large distances in $\bar{D}_{nn=3}$;
3: $\dot{d} \leftarrow \max\{\bar{D}_{nn=3} \setminus O\}$;
4: $\varepsilon_1 \leftarrow$ Randomly select a real number in $(0, 1)$;
5: $k \leftarrow (d^{max}/\dot{d}) * \sqrt{-\log(\varepsilon_1)}$;

In Algorithm 12, the mean distance, $\bar{d}_{nn}$, can be always calculated from the top-three or top-four nearest-neighbors. Actually, using top-three nearest-neighbors was applicable to most tested datasets in our experiments. Using a larger number of nearest neighbors did not significantly differ from using three or four, which is similar to using 4-dist for the estimation of the parameter *Eps* of DBSCAN. We used the Grubbs' test to detect the distances which are significantly larger than the other ones, and then we know the leftmost point where the curve tends to flatten (Figure 5-3). Furthermore, it is not necessary to sort all the distances when Grubbs' test is used, which implies less computation cost compared with the M. Ester method. We also noticed that Grubbs' test worked well for most datasets in the experiments by setting the significance level to 0.05.

The parameter r is used to help estimate the parameter ε_2 which is the threshold of the local density. From Definition 12, we know that such a threshold is directly related to the landmark-cardinalities. Thus, the expected average local density can be represented by the average value of the landmark-cardinalities. We use a positive real number r to represent a ratio of the average local density. Then, the threshold ε_2 can be estimated by the following Equation.

$$\varepsilon_2 = r \cdot \frac{\sum_{l \in \text{Ł}} card(l)}{g} \tag{5.2}$$

where g is the number of the generated landmarks. In most cases, the parameter r can be in the range of [0.5, 1.5]. Users are recommended to select a larger value for the parameter r if the density of outliers is higher, and vice versa.

Algorithm 13 The proposed clustering algorithm

Input: X, ξ, α, β, r

Output: P

1: Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two n-by-1 vectors, where n is the number of data points. $\mathcal{P}_1$ and $\mathcal{P}_2$ are used to distinguish whether a data point is an outlier nor not.
2: Ł $\leftarrow$ By the LAI-LBG algorithm with X, ξ and β;
3: $\mathcal{P}_1 \leftarrow$ Do the Grubbs' test for the landmark citations using α as the significance level. Mark the outliers by "zero", otherwise by "one";
4: $k, \varepsilon_1 \leftarrow$ Estimated by Algorithm 12;
5: $\varepsilon_2 \leftarrow$ Estimated by Equation (5.2);
6: $\mathcal{C} \leftarrow$ Call the modified FN-DBSCAN algorithm (Algorithm 11) with k, ε_1 and ε_2;
7: $\mathcal{P}_2 \leftarrow$ Label all data points according to the landmark cluster index, $\mathcal{C}$. A data point is marked by "zero", if it is cited by a landmark which is recognized as an "outlier" by the modified FN-DBSCAN;
8: $P \leftarrow \mathcal{P}_1 \circ \mathcal{P}_2$; // The operator "$\circ$" means the Hadamard product (elementwise product), i.e. $[\mathcal{P}_1 \circ \mathcal{P}_2]_i \leftarrow [\mathcal{P}_1]_i [\mathcal{P}_2]_i$, for all $1 \leq i \leq n$. The data points labelled by "zeros" are outliers.

5.1.3 A Further Discussion on Outliers Detection

The Grubbs' test in Step 2) can detect the outliers in a Voronoi region. However, sometimes, a landmark may be generated in the outliers data area if the density of outliers is very high or too much landmarks are generated when a very low value is

selected for the parameter ξ. Figure 5-2(b) shows an example of this case. We notice that the area near the landmark l has a lower density than the other areas since all the data points cited by l are outliers. Fortunately, the modified FN-DBSCAN algorithm can detect this low density area in a later step, because the landmark l itself will be considered as an outlier. Consequently, the data points cited by l are eventually considered as outliers.

Finally, we summarize the proposed clustering algorithm in Algorithm 13.

5.2 Experimental Results

In this section, we will evaluate the performance of the proposed clustering algorithm, i.e. Algorithm 13. In the following experiments, both clustering efficiency and clustering quality were tested. We compared the results of the proposed method with those of the original FN-DBSCAN algorithm and the landmark FN-DBSCAN algorithm. In addition, the results of the proposed method implemented with LAI-LBG were also compared with the other VQ implementations including LBG, AI-LBG (local) and AI-LBG (global). Five synthetic datasets were used. The details of these datasets are shown in Table 5.1 and illustrated in Figure 5-4, respectively. The clustering quality was evaluated by Rand-Index according to the true partitions (gold standards). All the results shown in this section were the average value of 10 times runs. In addition, the performances were evaluated with different number of landmarks.

Table 5.1: Datasets Used in Experiments.

Name	**Size**	**♯dimensions**	**♯clusters**	**outliers**
Banana	16437	2	2	no
Banana (outliers)	16532	2	2	yes
Anchor	28196	2	2	yes
Gaussian (3 clusters)	33000	2	3	no
Gaussian (14 clusters)	28000	2	14	no

The symbol ♯ means "the number of".

In the experiments, we observed that no matter what VQ implementation was applied, the proposed method definitely could run much faster than the original FN-DBSCAN algorithm when the same or higher clustering quality was achieved, which

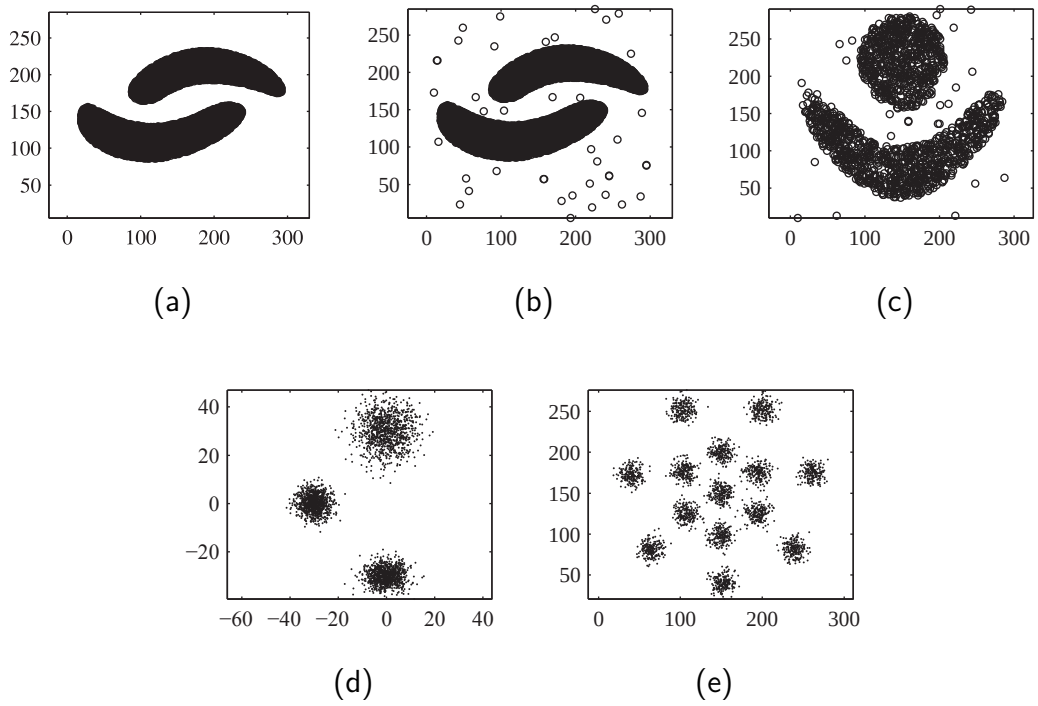


Figure 5-4: Five datasets used in our experiments. (a) The standard Banana dataset. (b) Banana dataset with outliers. (c) Anchor dataset. (d) Gaussian dataset (3 clusters). (e) Gaussian dataset (14 clusters).

indicated that the idea of using VQ to generate landmarks indeed significantly improved the clustering efficiency. Now, we detail the experimental results for each dataset as follows.

For the standard Banana dataset (Table 5.2), the proposed method with AI-LBG (global) achieved the best clustering quality because of the best vector quantization quality provided, but with its efficiency, it fell far behind other vector quantization implementations including LBG, AI-LBG and the proposed LAI-LBG. Even so, the efficiency was still much better than that of the original FN-DBSCAN algorithm when the number of landmarks was small (0.1561 second VS 7.8780 seconds with 15 landmarks). The clustering quality of using LAI-LBG was very close to that of using AI-LBG (global), but the efficiency was about 2.5 times higher than the latter one's when the number of landmarks was 15. We observed that the difference in the efficiency became larger when the number of landmarks increased, e.g. when

Table 5.2: The Clustering Results for the Banana dataset.

N[1]	Original FN-DBSCAN		Landmark[2] FN-DBSCAN		Proposed							
					LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time
15	1.0000	7.8780	0.5775	0.2180	0.7526	0.0507	0.9991	0.0704	1.0000	0.1561	1.0000	0.0621
20			0.5537	0.2190	0.8761	0.0701	0.9756	0.0781	1.0000	0.2164	0.9998	0.0800
25			0.5416	0.2340	0.9997	0.0878	0.9993	0.0819	0.9826	0.3083	0.9998	0.0995
50			0.5193	0.2340	0.9867	0.1677	0.9849	0.1913	0.9999	0.8540	0.9998	0.1638
100			0.5064	0.2490	0.9998	0.3335	0.9328	0.4486	0.9999	2.5644	0.9997	0.3179
200			0.5005	0.2960	0.9995	0.5946	0.9986	1.2480	0.9998	7.7005	0.9997	0.5771

[1]N indicates the number of landmarks.
[2]The landmark FN-DBSCAN algorithm started to achieve an acceptable Rand-Index value at 1.0000 since the number of landmarks was 413. The corresponding execution time was 0.3900s.

Table 5.3: The Clustering Results for the Banana (outliers) dataset.

N[1]	Original FN-DBSCAN		Landmark FN-DBSCAN		Proposed							
					LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time
15	0.9946	7.8630	0.6187	0.2180	0.7551	0.1716	0.8768	0.1772	0.9999	0.2867	0.9998	0.1871
20			0.6010	0.2190	0.9288	0.1814	0.9984	0.2144	0.9998	0.3432	0.9996	0.2009
25			0.5821	0.2190	0.9997	0.2085	0.9677	0.2086	0.9998	0.4329	0.9996	0.2224
50			0.5426	0.2340	0.9998	0.2534	0.9945	0.2731	0.9999	0.8970	0.9998	0.2476
100			0.5216	0.2650	0.9996	0.3571	0.9791	0.5186	0.9999	2.5779	0.9996	0.3880
200			0.9946	0.2970	0.9990	0.6589	0.9986	1.4020	0.9380	7.6400	0.9996	0.6064

[1]N indicates the number of landmarks.

the number of landmarks was 200, LAI-LBG achieved over 13 times faster than AI-LBG (global), whereas their qualities were almost the same. In addition, we noticed that the landmark FN-DBSCAN algorithm could not obtain a good result within 200 landmarks. It used 413 landmarks to successfully achieve Rand-Index value at 1.0000, but took 0.39 second as the cost, which was about 6 times slower compared with the proposed method with LAI-LBG (with 15 landmarks). The experimental results for the Banana (with outliers) datasets and Anchor datasets were quite similar to that for the standard Banana dataset. The results are shown in Table 5.3 and Table 5.4, respectively. The high scores of Rand-Index achieved for these three datasets indicated that the proposed method could successfully detect the arbitrary shaped clusters even outliers existed.

For the Gaussian dataset with three clusters (Table 5.5), the proposed method

Table 5.4: The Clustering Results for the Anchor dataset.

N[1]	Original FN-DBSCAN		Landmark[2] FN-DBSCAN		Proposed							
					LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time
15	0.9572	16.9850	0.5958	0.3590	0.8227	0.5325	0.9976	0.5636	0.9753	0.6923	0.9891	0.5516
20			0.5864	0.3740	0.9799	0.5656	0.9766	0.6164	0.9855	0.8407	0.9990	0.5970
25			0.5618	0.3750	0.9932	0.5871	0.9921	0.6124	0.9810	0.9634	0.9993	0.6279
50			0.5310	0.3740	0.9995	0.6105	0.9988	0.5887	0.9997	1.6048	0.9996	0.5229
100			0.5124	0.4060	0.9986	0.7159	0.9989	0.9086	0.9999	4.2434	0.9999	0.6319
200			0.5028	0.4840	0.9997	1.0199	0.9981	2.0961	0.9997	12.3260	0.9996	0.9125

[1]N indicates the number of landmarks.
[2]The landmark FN-DBSCAN algorithm started to achieve an acceptable Rand-Index value at 0.9572 since the number of landmarks was 961. The corresponding execution time was 0.8740s.

with AI-LBG (global) and with LAI-LBG achieved perfect clustering qualities. The efficiency of using LAI-LBG was over 680 times higher than that of the original FN-DBSCAN algorithm and from twice to five times higher than that of using AI-LBG (global). Using LBG or AI-LBG (local) as the VQ implementation in the proposed method failed to obtain such good results on quality, but they were still acceptable compared with the original FN-DBSCAN algorithm and the landmark FN-DBSCAN algorithm. During the experiment, we noticed that there was a special case for the landmark FN-DBSCAN algorithm, i.e. a very small number of landmarks to represent the input data distribution could achieve a good quality and efficiency, because one or two landmarks might be enough to well represent a Gaussian distributed cluster if the landmarks were near the cluster centers. However, since all three clusters in this dataset were large, it was difficult for the landmark FN-DBSCAN algorithm to precisely generate landmarks close to the cluster centers owing to the input order problem discussed in Section 4.1.1. In this experiment, the landmark FN-DBSCAN used 10 landmarks to achieve 0.9004 Rand-Index value, but the good result could not be kept with the increase of the number of landmarks. The quality started to be stable when the number of landmarks was more than 226, but the time consumption was 0.5460 second which was too much compared to that of the proposed method with LAI-LBG.

For the Gaussian dataset with 14 clusters (Table 5.6), similar to the previous results, the proposed method with LAI-LBG obtained the best combination of clustering

Table 5.5: The Clustering Results for Gaussian (3 clusters) dataset.

N[1]	Original FN-DBSCAN		Landmark[2] FN-DBSCAN		Proposed							
					LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time
10			0.9004	0.4210	0.9721	0.0351	0.9983	0.0605	1.0000	0.1619	1.0000	0.0605
15			0.8524	0.4210	0.9999	0.0467	0.9998	0.0741	1.0000	0.3061	1.0000	0.0896
20	0.9999	41.2620	0.8215	0.4210	0.9999	0.0720	0.9999	0.0995	1.0000	0.4289	1.0000	0.1130
25			0.7934	0.4360	0.9944	0.0954	0.9999	0.1230	1.0000	0.5850	1.0000	0.1308
30			0.7760	0.4370	1.0000	0.1073	0.9998	0.1500	1.0000	0.7565	1.0000	0.1540

[1]N indicates the number of landmarks.
[2]The landmark FN-DBSCAN algorithm started to achieve an acceptable Rand-Index value at 0.9991 since the number of landmarks was 226. The corresponding execution time was 0.5460s.

Table 5.6: The Clustering Results for Gaussian (14 clusters) dataset.

N[1]	Original FN-DBSCAN		Landmark[2] FN-DBSCAN		Proposed							
					LBG		AI-LBG (local)		AI-LBG (global)		LAI-LBG	
	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time	Rand	Time
14[3]			0.9983	0.3740	0.0714	0.0546	0.0714	0.0858	0.0714	0.2516	0.0714	0.0780
50			0.9641	0.4210	0.5153	0.1129	0.9976	0.2964	0.9998	1.4100	0.9997	0.1892
60	0.9988	20.7320	0.9607	0.4680	0.7371	0.1404	0.9982	0.3606	0.9998	1.8351	0.9997	0.2109
70			0.9576	0.4840	0.5777	0.1873	0.9984	0.4251	0.9998	2.3869	0.9998	0.2421
80			0.9549	0.3900	0.8964	0.2165	0.9984	0.5031	0.9998	2.9837	0.9997	0.2963

[1]N indicates the number of landmarks.
[2]The landmark FN-DBSCAN algorithm started to achieve an acceptable Rand-Index value at 0.9929 since the number of landmarks was 2850. The corresponding execution time was 2.1840s.
[3]14 landmarks was an special setting for the landmark FN-DBSCAN algorithm.

quality and efficiency. Since the clusters were small, it was easier for the landmark FN-DBSCAN algorithm to generate landmarks near the cluster centers. We found the landmark FN-DBSCAN could only use 14 landmarks, which was the exact number of the clusters, to represent the input datasets. Unfortunately, with the efficiency, it still fell behind that of the proposed method with LAI-LBG. In addition, we noticed that using LBG as the implementation resulted in a very poor clustering quality. The main reason was that the tested dataset contained a considerable number of clusters. In this case, it was difficult to initialize a balanced distribution for landmarks by the random selection method, which in the later stage highly influenced the quantization result for LBG.

Next, we tested the performance of the improved landmark FN-DBSCAN with a real-world dataset, the Optical Recognition of Handwritten Digits (optdigits) dataset (Fig. 5-5), selected from the UCI machine learning database (https://archive.ics.uci.

edu/ml/datasets/). The optdigits dataset contained 10 classes of handwritten digits from 43 people. The total number of the data points was 5,620 and the data dimension

Figure 5-5: The Optical Recognition of Handwritten Digits (optdigits) dataset.

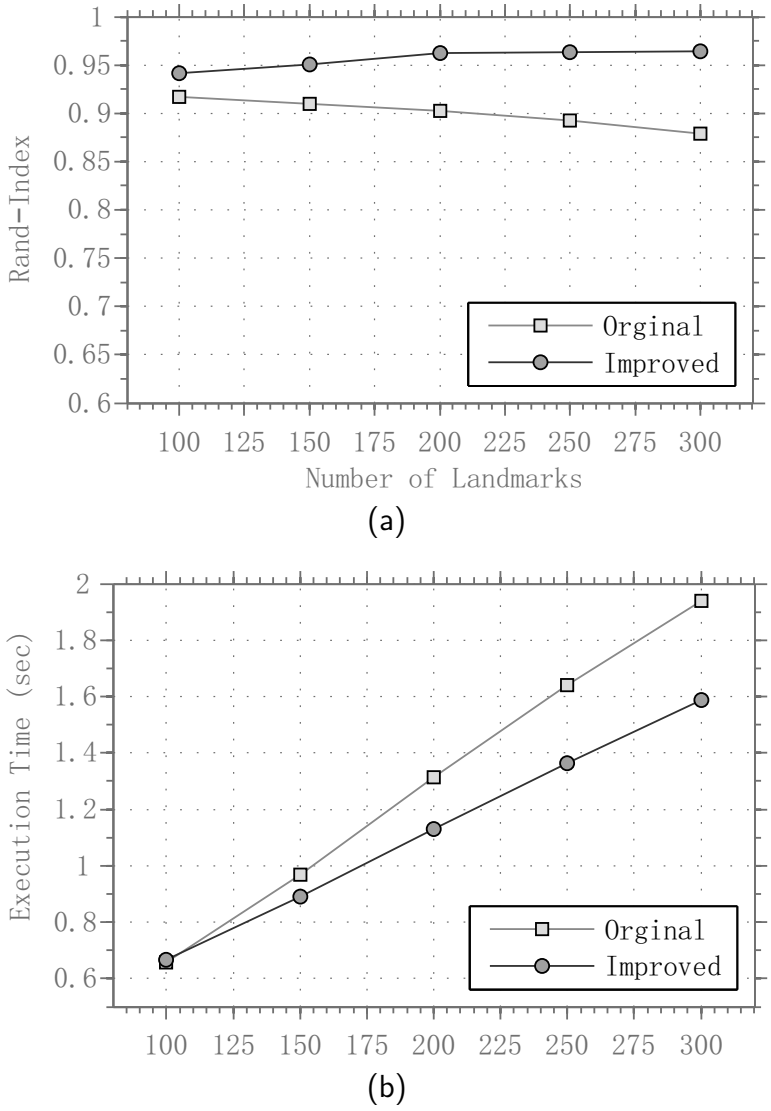


Figure 5-6: The results of clustering quality (a) and efficiency (b) for optdigits dataset regarding different number of landmarks.

was 64. The experimental results were compared with those of the landmark FN-DBSCAN algorithm. Here, we must note that there were two reasons for applying a post-processing to improve the clustering quality. One was that both of the two algorithms might label data points as noise, but the actual dataset did not contain any noisy data. Another reason was that the two algorithms might find more than 10 clusters in their results. Therefore, in the post-processing for the result of each algorithm, we only accepted the top-10 largest clusters, if the number of clusters was larger than 10, and reassigned the data points in the remaining smaller clusters, as well as the data points labelled as noise, to the closest accepted clusters.

The results for the clustering quality and efficiency with regard to different numbers of landmarks are shown in Fig. 5-6(a) and Fig. 5-6(b), respectively. We observed that the landmark FN-DBSCAN could not obtain a good clustering quality within 300 landmarks. Actually, the quality results, which the landmark FN-DBSCAN algorithm achieved, were not stable, which even decreased with the increasing number of the generated landmarks. The main reason was that 300 landmarks were not sufficient to represent the distribution of the optdigits dataset, because the data dimension was considerably high. On the other hand, we could see that the improved algorithm could continuously obtain a good clustering result when the number of landmarks varied from 100 to 300, and had a higher efficiency than the original landmark FN-DBSCAN algorithm.

5.3 Summary

In this chapter, we proposed a novel density-based clustering algorithm which is an improvement of the previously proposed landmark FN-DBSCAN algorithm, benefiting from the new features of LAI-LBG algorithm. The experiments presented in this chapter show that the proposed clustering algorithm achieved a similar high clustering quality to and a higher efficiency than the other techniques including the original FN-DBSCAN, landmark FN-DBSCAN and the proposed method with other vector quantization approaches.

Chapter 6

Incremental Clustering by Self-organizing Neural Network

As discussed in Chapter 2.2.2, using self-organizing neural networks, especially for the ones with a dynamic structure, is a more natural way to implement incremental clustering. However, recently, a much more difficult task for on-line learning is to efficiently and robustly learn data from noisy environments. In this chapter, we propose a novel method, called Enhanced Incremental Growing Neural Gas (Hi-GNG) as a solution for this problem. The rest of this chapter is organized as follows. In Section 6.2, we first review the Competitive Hebbian Learning (CHL) method, and then propose the enhanced CHL method. In Section 6.3, we propose the Hi-GNG algorithm. In Section 6.4, experimental results are presented. Finally, we summarize the features of Hi-GNG and give conclusions in Section 6.5.

6.1 Motivations of the Hi-GNG algorithm

Recently, with the increasing number and dimensions of data, the learning algorithms are required to efficiently deal with large number of signals. Moreover, a much more difficult task for on-line learning is to efficiently and robustly learn data from the distributions in which noisy data exist. The main difficulty is that learning algorithms have no prior knowledge of the whole data distribution. Thus, upon the arrival of

the first several data of a certain distribution, the amount of data is not sufficient to represent the whole distribution. At this time, learning algorithms cannot judge whether these data are noisy or normal. Consequently, for each iteration, the existing techniques, e.g. SOM, NG, and etc., have to respond to the new data and update the weight vectors of the corresponding neurons, which usually cause a critical deviation of the topology reflection in the result.

Another problem is that in most 'growing-type' self-organizing neural networks, such as GNG, the number of neurons will continuously increase owing to the growth strategy. The large number of neurons increases the computational cost of searching for the winner neurons in each iteration, which makes the training procedure inefficient. In fact, such a number cannot increase indefinitely because of the limited computer resources, such as CPU speed and memory size. To partially solve this problem, the maximum number of neurons in the network is usually predefined by an additional parameter. However, this yields another problem of determining the value of this parameter. If the parameter is set with a low value, some small clusters will be merged, indicating that the clustering result will be wrong. If the parameter is set too high, many neurons will be generated and some large cluster may be divided into several smaller clusters, which may also results in a low clustering quality.

6.2 Enhanced Competitive Hebbian Learning

The Competitive Hebbian Learning (CHL) method is a foundation of many competitive learning system. Here, we extend the conventional CHL to model the creation and elimination of connections between each pair of neurons, especially for the adjustment of the connection-strength of each connection.

6.2.1 Competitive Hebbian Learning

The classical Hebbian theory (also called Hebbian rule) [104] describes a basic mechanism for how neurons can connect with each other, more precisely, it shows how to create the connections between neurons. According to Hebbian theory, any two

neurons sufficiently near to each other that are repeatedly activated at the same time will tend to become 'associated' in the sense that a connection can be created between them.

CHL can be considered as an extension of the classical Hebbian theory in competitive learning systems. The key idea of CHL is that given an input signal, we find the nearest neuron and the second-nearest neuron in the network, after which we create a connection (edge) between these two neurons if such a connection does not already exist.

CHL is a good method for the creation of connections between neurons in competitive learning systems. However, CHL only provides a way of creating connections, but does not address the elimination of the existing connections. Some learning algorithms, such as GNG, use a local variable to record the 'age' of a connection. The 'age' of a connection will increase according to some updating strategy. If a connection's 'age' has exceeded a particular threshold, it will accordingly be removed. In general, such a threshold is a positive integer, e.g. the threshold is given by the parameter α_{max} in the GNG algorithm. Here, α_{max} is usually set with a big value because the network will not grow if α_{max} is too small. Unfortunately, considering that the given dataset contains some noisy data, the final graph will be significantly influenced by the noisy data if α_{max} is big, because the local variable 'age' may be easily reset to zero and the corresponding connections will be kept for a long time. This means that the learning algorithm will not robustly reflect the data relationships from noisy data distributions.

6.2.2 Enhanced Competitive Hebbian Learning

In order to solve this problem, we propose a new technique to generate topological structures, called enhanced competitive Hebbian learning (enhanced CHL). The enhanced CHL method considers both the creation of connections between neurons and the elimination of the existing connections between neurons.

The aim of enhanced CHL is to generate an undirected weighted graph, $\langle G, w\rangle$, where $G = \langle V, E, \psi_G\rangle$ consists of a set of vertices V, a set of edges E and an inci-

dence function ϕ_G. The weighting $w : E \to \mathbb{R}$ can be considered as a vector whose coordinates are indexed by the edge set E. For example, let $\{v_1, v_2\}$ be an unordered pair of two vertices of G and e be an edge of G. Then $\psi_G(e) = \{v_1, v_2\}$ indicates that $\{v_1, v_2\}$ is associated with e, and $w(e)$ represents the weight of the edge e.

In this dissertation, the vertex and the weighted graph are called a *neuron* and a *network*, respectively, and are denoted by n and net, respectively. On the basis of these standard concepts of graph theory, we give the formal definition of a connection in the enhanced CHL method.

Definition 13 *(connection) Let n_1 and n_2 be two neurons in the network ($n_1 \neq n_2$) and let $\{n_1, n_2\}$ be the edge between n_1 and n_2. A connection, denoted as $c(n_1, n_2)$, is a pair:*

$$c(n_1, n_2) = \langle \{n_1, n_2\}, s \rangle \tag{6.1}$$

where s is a non-negative integer, called the *connection-strength* of $c(n_1, n_2)$. For convenience, we also call it the *connection-strength* between n_1 and n_2, denoted by $s(n_1, n_2)$.

Given an input signal ξ and a network, net, the nearest neuron, $n^\star$, and the second-nearest neuron, $n^{\star\star}$, to the signal ξ can be obtained by Equation (6.2).

$$\begin{aligned} n^\star &= \arg\min_{n_i \in N(net)} \|\xi - W_{n_i}\| \\ n^{\star\star} &= \arg\min_{n_i \in N(net) \setminus \{n^\star\}} \|\xi - W_{n_i}\| \end{aligned} \tag{6.2}$$

where W_{n_i} is the weight vector of n_i and $N(net)$ is the neuron set of the network.

Then let net be a network and $n^\star$, $n^{\star\star}$ be the nearest neuron and the second-nearest neuron to an input signal ξ, respectively ($n^\star, n^{\star\star} \in N(net)$). The enhanced CHL method is described in the following four parts:

1. CREATION: Create a new *connection* $c(n^\star, n^{\star\star})$ if $c(n^\star, n^{\star\star})$ does not exist. When the new *connection* is established, the *connection-strength* is initialized by: $s(n^\star, n^{\star\star}) \leftarrow 1$.

2. UPDATING: If the *connection*, $c(n^\star, n^{\star\star})$, already exists, then its *connection-strength* is enhanced by: $s(n^\star, n^{\star\star}) \leftarrow s(n^\star, n^{\star\star}) + \Delta s$, where Δs is a positive real number. If $\exists c(n^\star, n_i)$ where $n_i \in N(net)$ such that $n_i \neq n^{\star\star}$, then the *connection-strength* of these *connections* are reduced by: $s(n^\star, n_i) \leftarrow s(n^\star, n_i) - \nabla s$, where ∇s is a positive real number.

3. DECAYING: when the elapsed time is an integer multiple of some desired number, the *connection-strength* of every *connection* in the network will be reduced by: $s(n_i, n_j) \leftarrow s(n_i, n_j) - \nabla s, n_i, n_j \in N(net)$.

4. ELIMINATION: if $\exists c(n_i, n_j)$ where $n_i, n_j \in N(net), n_i \neq n_j$ such that $s(n_i, n_j) = 0$, then remove the *connection*, $c(n_i, n_j)$.

In the creation part, the enhanced CHL method follows the idea of Hebbian theory, i.e. a *connection* is created between two neurons if they are near enough and are activated at the same time. However, in our model, we do not care only about how to create a *connection*, but also about the *connection-strength* of the created *connection*. The *connection-strength* of the newly created *connections* are initialized by one, indicating that these *connections* are temporal ones. If they are not enhanced in the following few steps, they can be quickly removed.

In the second part, the updating is also related to Hebbian theory. If a *connection* between two *neurons* has already been created in some earlier steps and subsequently the two *neurons* are again activated at the same time, then the *connection-strength* will be enhanced. Moreover, we also consider the opposite situation: if the two neurons are not simultaneously activated, the *connection-strength* between them will be weakened.

In the decaying part, the *connection-strength* between neurons will be gradually weakened with time. In order to make an easy calculation, the *connection-strength* will be reduced if the total elapsed time is an integer multiple of some desired number.

In the eliminating part, the *connections* in the *network* will be removed if their *connection-strength* values are zero.

Like CHL, the enhanced CHL method is also a pure topology generating method.

To incrementally learn the given data distribution, $p(\xi)$, the learning algorithm also requires a way of inserting neurons and a vector quantization technique in order to generate and place the neurons in regions in which the probability density $P(\xi) > 0$. In addition, we must note that it is possible to specify the increment and decrement of the *connection-strength*, i.e. Δs and ∇s, when the enhanced CHL method is applied in a specific learning algorithm. An example will be presented in Section 6.3.

6.3 The Algorithm of Enhanced Incremental Growing Neural Gas

In this section, we present the enhanced incremental growing neural gas (Hi-GNG) algorithm which is based on the enhanced CHL method.

In Hi-GNG, the classical SOM neuron model is modified by assigning an additional local attribute called a *maturity-level*. Now we give the definition of a *neuron* in the Hi-GNG algorithm.

Definition 14 *(neuron) A neuron, denoted by n, is a pair:*

$$n = \langle W, m \rangle \tag{6.3}$$

where W is the weight-vector of n in the input space and m is a non-negative integer called the *maturity-level*. We denote the weight of n and the *maturity-level* of n by W_n and $m(n)$, respectively.

In this modified neuron mode, the new attribute m functions as a local counter to count the number of times that a *neuron* becomes the nearest neuron or the second-nearest neuron. Given an integer number, $\hat{m}$, as the threshold, if the *maturity-level* of *neuron* n is greater than $\hat{m}$, i.e. $m(n) > \hat{m}$, we say n becomes 'mature'.

On the basis of this modified neuron model and the enhanced CHL method presented in Section 6.2.2, we present the Hi-GNG algorithm as follows:

1. Initialize the network with an empty graph.

2. Randomly generate a signal ξ from $P(\xi)$.

3. If the network is empty or the Euclidean distance between the input signal and the nearest (or the second-nearest) neuron satisfies Equation (6.4), then insert two *neurons*.

$$\exists n \in \{n^\star, n^{\star\star}\},\ \|\xi - W_n\| > \sigma \tag{6.4}$$

   where ξ is the current input signal and W_n is the weight vector of the nearest neuron ($n^\star$) or the second-nearest neuron ($n^{\star\star}$).

   Let n_1 and n_2 be the two newly inserted neurons. The weight of n_1 is set with $W_{n_1} \leftarrow \xi$ and n_2 is placed very close to n_1 in the input space. Here, W_{n_2} is randomly generated, such that $\|W_{n_1} - W_{n_2}\| < \frac{\sigma}{10}$. Then, create a *connection* between n_1 and n_2 with the *connection-strength*: $s(n_1, n_2) \leftarrow 1$. Finally, the adaption of the current input signal, ξ, is complete. Continue to adapt the next signal.

4. Update the weight vectors of the neurons using Equation (6.5).

$$W_n = \begin{cases} W_n + \epsilon_b (\xi - W_n), & \text{if } n = n^\star \\ W_n + \epsilon_n (\xi - W_n), & \text{if } \exists c(n, n^\star) \end{cases} \tag{6.5}$$

   where $n^\star$ is the nearest neuron to the input signal ξ, and ϵ_b and ϵ_n are two real numbers in the range of $(0, 1)$, which represent the constant learning rate for the nearest neuron and its topological neighbors. In general cases, ϵ_b is greater than ϵ_n.

5. Increment the *maturity-level* of the nearest neuron, $n^\star$, and the second-nearest neuron, $n^{\star\star}$.

6. Create a *connection* between $n^\star$ and $n^{\star\star}$ if there is no such a *connection*. Initialize the *connection-strength* by the *maturity-level* of the nearest neuron, $n^\star$.

7. Increment the *connection-strength* using Equation (6.6), if a connection exists between $n^\star$ and $n^{\star\star}$.

$$s(n^\star, n^{\star\star}) \leftarrow s(n^\star, n^{\star\star}) + \Delta s \tag{6.6}$$

where Δs is a positive integer presenting the increment of the *connection-strength*. In general, Δs can be appropriately calculated according to the *maturity-level* of the nearest neuron, $n^\star$.

$$\Delta s \leftarrow m(n^\star) \tag{6.7}$$

8. Reduce the *connection-strength* between the nearest neuron, $n^\star$, and its topological neighbors using Equation (6.8).

$$s(n^\star, n_i) \leftarrow s(n^\star, n_i) - \nabla s, \text{ if } \exists c(n^\star, n_i) \tag{6.8}$$

where ∇s can be calculated according to the *maturity-level* of $n^\star$, given by Equation (6.9).

$$\nabla s \leftarrow \log(m(n^\star)) \tag{6.9}$$

9. Remove the *connections* whose *connection-strength* is zero and if this isolates some *neurons*, also remove the isolated *neurons* as well.

10. Reduce the *connection-strength* of all the *connections* in the network using Equation (6.10), if the number of iterations so far is an integer multiple of the parameter α.

$$s(n^\star, n_i) \leftarrow s(n^\star, n_i) - \nabla s, n_i \in N(net) \tag{6.10}$$

where ∇s can be $\log(m(n^\star))$ according to Equation (6.9).

11. If the stopping condition is not met, then go to step 2). Otherwise, output the network which only consists of the *neurons* whose *maturity-levels* are larger than the parameter $\hat{m}$

Algorithm 14 The Hi-GNG Algorithm

Input: X, σ, ϵ_b, ϵ_n, α, $\hat{m}$

Output: an undirected weighted graph only consists of the mature neurons ($m(n) > \hat{m}$) and their corresponding connections.

Initialize the network with an empty graph;
$g \leftarrow 0$;
repeat
  $\xi \leftarrow$ generate a signal ξ from $P(\xi)$ randomly;
  Search the network for the nearest neuron, $n^\star$ and the second-nearest neuron, $n^{\star\star}$ by:
  $n^\star \leftarrow \arg\min_{n_i \in N(net)} \|\xi - W_{n_i}\|$
  $n^{\star\star} \leftarrow \arg\min_{n_i \in N(net)\setminus\{n^\star\}} \|\xi - W_{n_i}\|$
  if $N(net) = \phi$ or $\|\xi - W_{n^\star}\| > \sigma$ or $\|\xi - W_{n^{\star\star}}\| > \sigma$ **then**
    Insert two neurons, n_1 and n_2, to the network and set them by:
    $W_{n_1} \leftarrow \xi$ and randomly generate W_{n_2}, such that $\|W_{n_1} - W_{n_2}\| < \frac{\sigma}{10}$. Then create a connection between n_1 and n_2 and set the connection-strength by: $s(n_1, n_2) \leftarrow 1$;
    continue;
  end if
  Update the neurons by:
  $W_{n^\star} \leftarrow W_{n^\star} + \epsilon_b (\xi - W_{n^\star})$
  $W_{n_i} \leftarrow W_{n_i} + \epsilon_n (\xi - W_{n_i})$ // $\exists c(n_i, n^\star)$
  Increment the maturity levels of $n^\star$ and $n^{\star\star}$ by:
  $m(n^\star) \leftarrow m(n^\star) + 1$
  $m(n^{\star\star}) \leftarrow m(n^{\star\star}) + 1$
  if $c(n^\star, n^{\star\star})$ already exists **then**
    Update the connection-strength of $c(n^\star, n^{\star\star})$ by:
    $s(n^\star, n^{\star\star}) \leftarrow s(n^\star, n^{\star\star}) + m(n^\star)$;
  else
    Create a connection $c(n^\star, n^{\star\star})$ and set the connection strength as:
    $s(n^\star, n^{\star\star}) \leftarrow m(n^\star)$;
  end if
  Reduce the connection-strength of the connections between $n^\star$ and its topological neighbors by:
  $s(n^\star, n_i) \leftarrow s(n^\star, n_i) - \log(m(n^\star))$;
  Remove the connections whose connection-strength is zero and if this makes some neurons isolated, also remove the isolated neurons.
  if $g \mod \alpha = 0$ **then**
    Reduce the connection-strength of all existing connections by:
    $s(n_i, n_j) \leftarrow s(n_i, n_j) - \log(m(n^\star))$;
  end if
  $g \leftarrow g + 1$;
until the stopping condition is reached

In step 3), as with the IGNG algorithm, the Hi-GNG algorithm also applies a distance-check strategy when neurons are inserted. However, the IGNG algorithm inserts only one neuron, whereas the Hi-GNG algorithm inserts two neurons. The main reason for inserting two neurons is that the distance between the input signal and the nearest neuron is larger than a threshold, which means that the current input signal is far away from the signals learned so far. Thus, there is a high probability that the current signal belongs to a new cluster or is a noisy signal. In this case, the network should create a 'new land' (a new topological structure disjoint from the existing structure) to represent the new cluster or the noisy signal (this can also be regarded as a small cluster). If we consider that each cluster is represented by a component in the network graph, two connected neurons can be inserted to create a minimal component for representing a new cluster. A component which contains only one isolated neuron will be directly removed in the next iteration because of step 9).

In step 5), the *maturity-level* is incremented, but note that only the nearest neuron and the second-nearest neuron will be manipulated, which is different from other techniques, such as IGNG. The IGNG algorithm uses a local variable, called 'age', to identify whether or not a neuron is 'mature'. The nearest neuron and all its topological neighbors will be manipulated in the IGNG algorithm.

From step 6) to step 10), the enhanced CHL method is applied. When a *connection* is created, the *connection-strength* is very weak, which means that the new *connection* must be enhanced within a small number of iterations, or it will be removed quickly owing to step 9). This strategy makes the Hi-GNG algorithm robust to noisy signals. When a noisy signal is input, there is a high possibility that Equation (6.4) is satisfied. Then, the algorithm inserts two connected neurons to respond to this noisy signal. Because the noisy signal can be expected to have a low probability of being regenerated, it is also expected that the *connection* between the just created two neurons is not easily enhanced. Therefore, the neurons presenting the noisy data can be quickly removed. This is similar to the short-term plasticity in biological neural behaviors. On the other hand, the *connections* between those neurons which present the normal data can be repeatly enhanced. Such neurons can therefore be kept in

the network for a long time. This appears like the long-term plasticity in biological neural behaviors.

Details regarding the implementation of Hi-GNG are presented in Algorithm 14.

6.4 Experimental Results

6.4.1 Batch Learning

n the following experiments, we focus on the batch learning experiments. Two artificial datasets were used to test the performance of Hi-GNG compared with that of the GNG algorithm. We summarized the information regarding these artificial datasets as follows:

1. Dataset 1 contained 1386 two-dimensional data in four clusters. The cluster at the top of Fig. 6-1(a) was represented by a twisty shape and another three smaller clusters were located under the former bigger one. No noisy data existed in this dataset.

2. Dataset 2 had two banana shaped clusters, as illustrated in Fig. 6-1(b). In total, there were 16532 data including some noisy data.

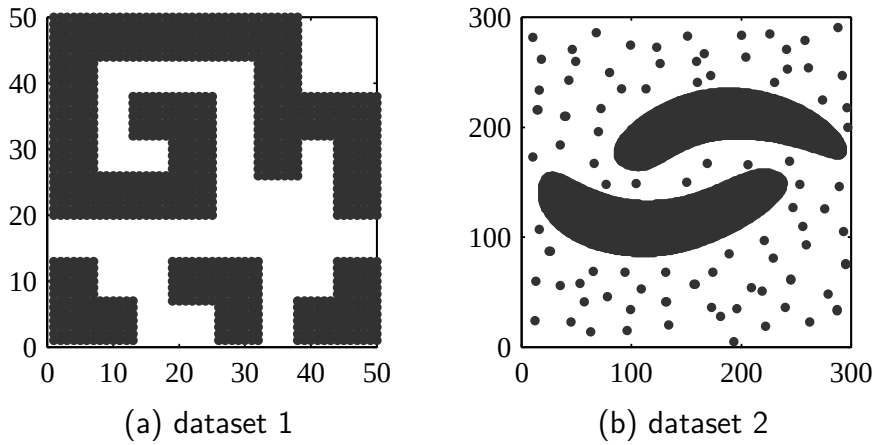


Figure 6-1: Artificial datasets used in the experiments. (a) dataset 1. (b) dataset 2.

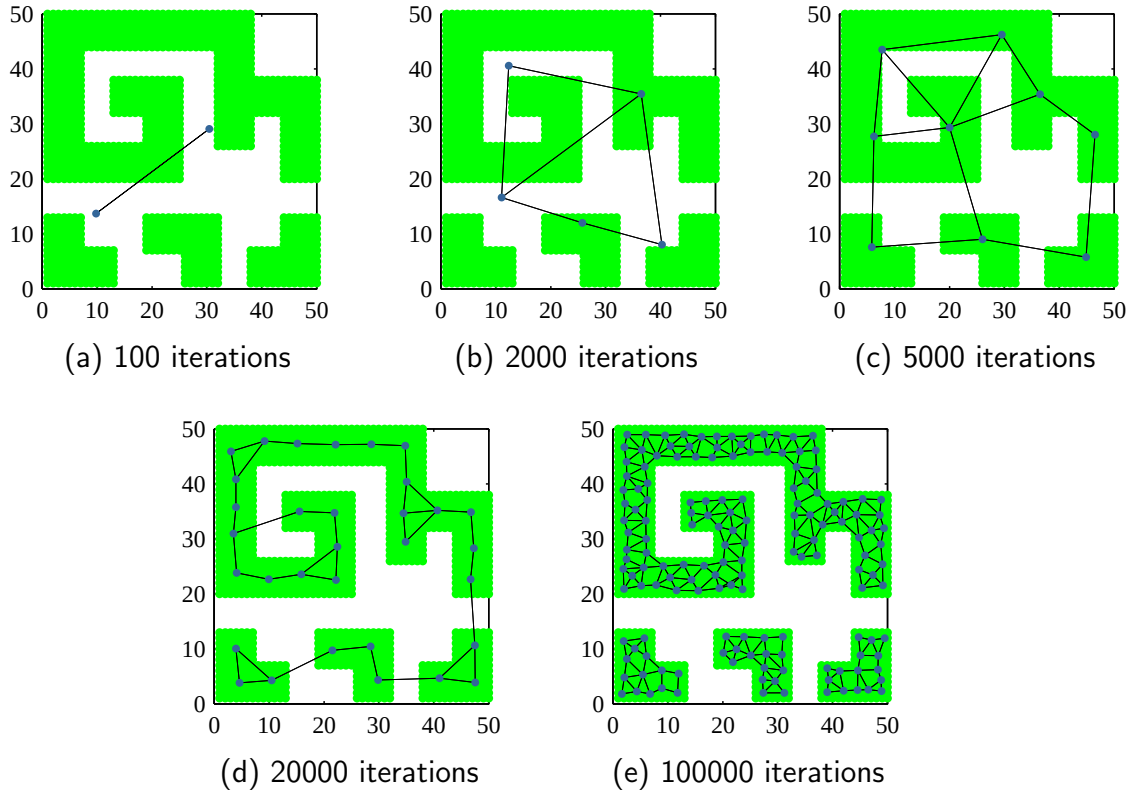


Figure 6-2: The snapshots in the learning procedure for dataset 1 with GNG. The parameters were selected as follows: $\lambda = 650$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$, $\alpha = 0.5$, $d = 0.0005$.

In the experiment involving dataset 1, we captured five snapshots during the learning procedure for both GNG (Fig. 6-2) and Hi-GNG (Fig. 6-3). We observed that the Hi-GNG algorithm generated neurons faster than the GNG algorithm. At the beginning, Hi-GNG started with an empty graph and inserted many neurons very quickly, and the newly inserted neurons started as immature neurons (represented by the 'diamond' shaped markers in Fig. 6-3(a)), whereas the GNG algorithm started with two neurons and inserted neurons slowly (Fig. 6-2(a) - Fig. 6-2(c)). We also noticed that in the Hi-GNG learning procedure, immature neurons were widely placed according to the input distribution. In each iteration, new neurons were directly inserted near the input data if the distance-check (by Equation (6.4)) was satisfied. About 1500-2000 iterations later, some immature neurons became mature (represented by 'solid circle' shaped markers in Fig. 6-3(b)), and when the number of iteration was 5000, most im-

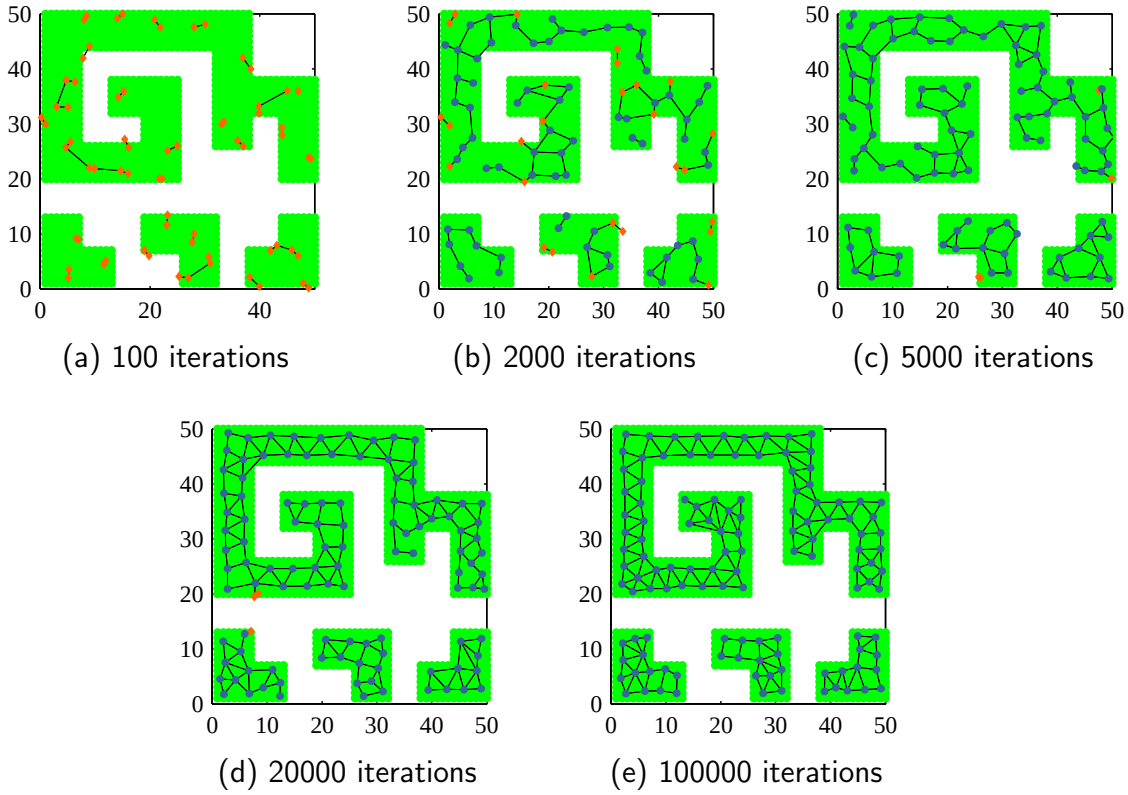


Figure 6-3: The snapshots in the learning procedure for dataset 1 with Hi-GNG. The parameters were selected as follows: $\sigma = 5$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\hat{m} = 30$, $\alpha = 80$.

mature neurons became mature and more connections were established (Fig. 6-3(c)). When the number of iterations was more than 20000, the data distribution was well adapted by Hi-GNG (Fig. 6-3(d)) comparing with that of the GNG algorithm, which still needed more iterations to finish the adaption (Fig. 6-2(d)). Although we could choose a smaller value for the parameter λ in GNG, a smaller value may sometimes result in a bad adaption result.

In addition, we also observed that the number of neurons generated by these two algorithms was significantly different during the whole learning procedure (shown in Fig. 6-6(a)). If we had not provided an upper limit for the number of neurons, the insertion of neurons would have been a never-ending operation in the GNG algorithm. On the other hand, the number of neurons in Hi-GNG stopped increasing at around 120 after 45000 iterations. Moreover, we separately recorded the number of mature

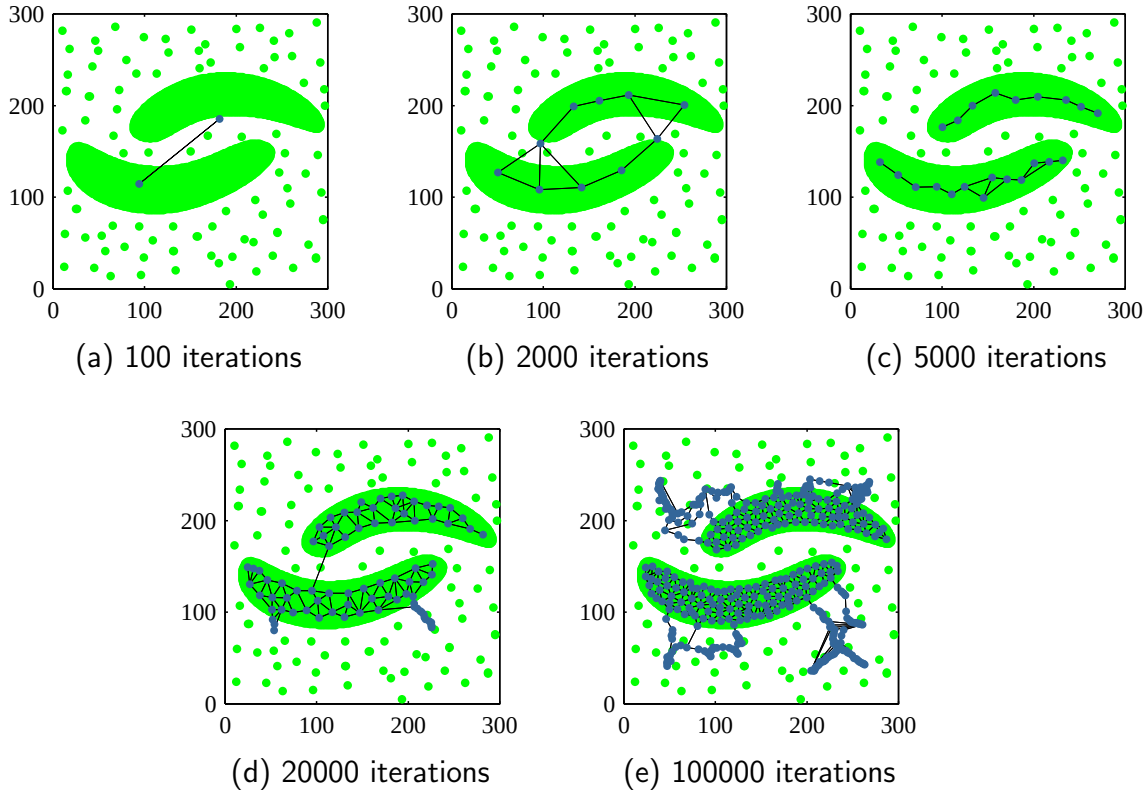


Figure 6-4: The snapshots in the learning procedure for dataset 2 with GNG. The parameters were selected as follows: $\lambda = 250$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$, $\alpha = 0.5$, $d = 0.0005$.

neurons and immature neurons. The number of mature neurons started from zero and slowly increased at the beginning of several hundreds of iterations, but started to increase very quickly afterwards. Two or three thousands of iterations later, the rate of increase slowed again. Finally, the number became stable when it was at around 120. The number of immature neurons also started at zero, but on the contrary, it first increased quickly and then also decreased quickly. Finally, the number became stable around zero or two and there were very few fluctuations.

We also captured five snapshots of the experiment regarding dataset 2 for both GNG (Fig. 6-4) and Hi-GNG (Fig. 6-5). First, we observed that Hi-GNG could generate neurons faster than GNG, which was similar to the previous experiment with dataset 1. However, because dataset 2 contains noisy data, the topological structures generated by these two algorithms were quite different. The topological structure

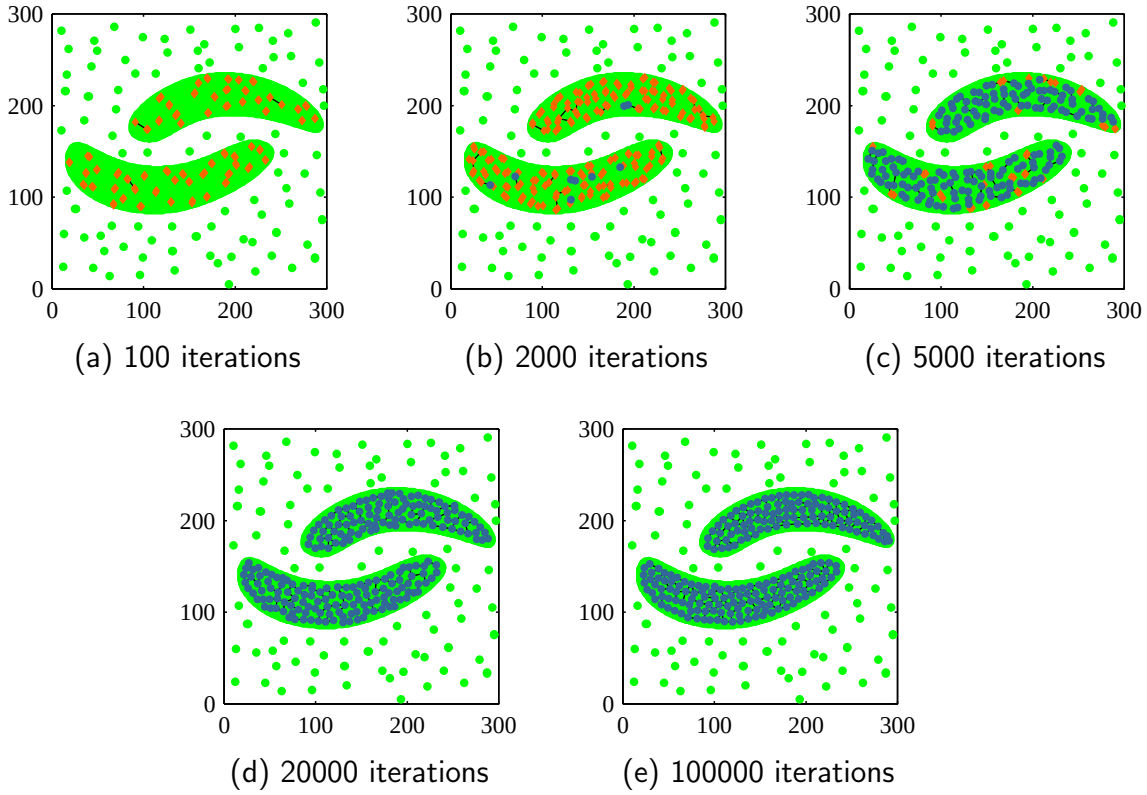
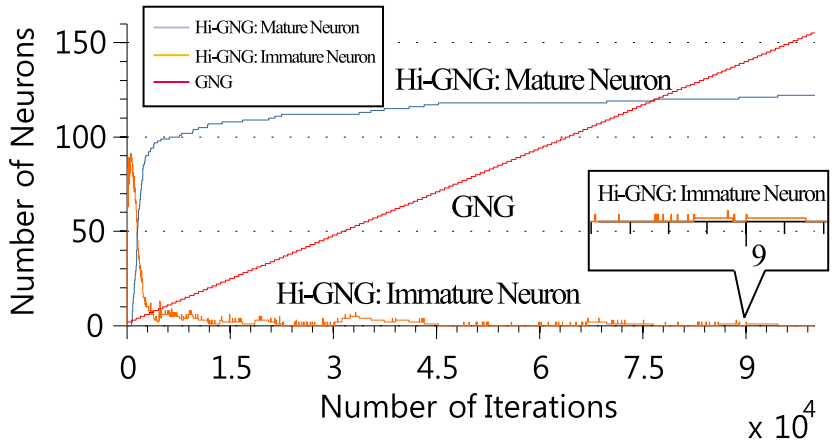
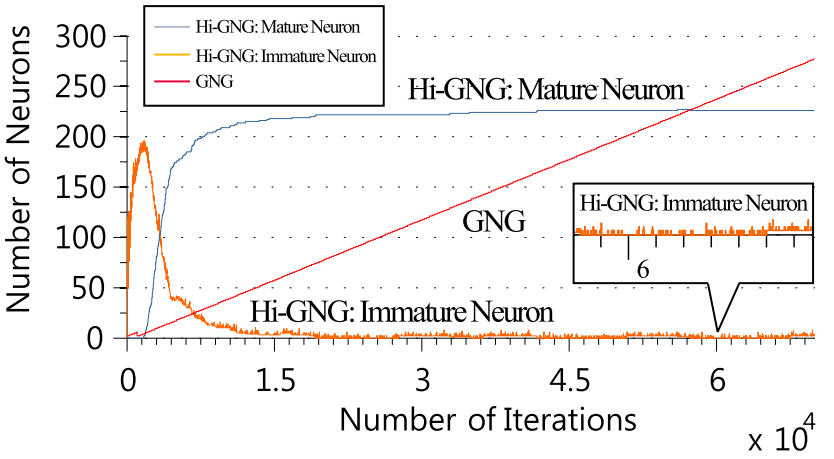


Figure 6-5: The snapshots in the learning procedure for dataset 2 with Hi-GNG. The parameters were selected as follows: $\sigma = 10$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\hat{m} = 30$, $\alpha = 50$.

generated by GNG became twisty and distorted after 20,000 iterations (Fig. 6-2(d)), and continuously deteriorated with the increasing number of iterations (Fig. 6-2(e)). The reason was that the noisy data were usually far away from the normal data, which resulted in a large distance value being calculated using $\|\xi - W\|^2$. Then, this large distance significantly affected the updating of the weights. On the other hand, the topological structure generated by Hi-GNG was stable and always reflected the distribution of the normal data. We recorded the number of neurons to help explain why these results were obtained (Fig. 6-6). Like the previous result for dataset 1, the number of mature neurons in Hi-GNG became stable when the number of iterations was more than 20,000. However, at the same time, the number of immature neurons was not so stable and frequently changed. This unstable number appeared when Hi-GNG responded to the noisy data; new immature neurons were inserted into the



(a) The result for dataset 1



(b) The result for dataset 2

Figure 6-6: The number of neurons during the learning procedures.

network, because the large distance between the noisy data and the nearest neuron satisfied the distance-check (Equation (6.4)). Since the connection-strength between the two newly inserted immature neurons was very weak and not easily enhanced, it easily decreased to zero after a few numbers of iterations, which led to the removal of the newly inserted neurons. As a result, the mature neurons which represented the normal data were kept and those immature neurons which represented the noisy data were generated and removed quickly. To better understand this experiment, we have uploaded a video demonstration of this learning procedure on the YouTube website, which can be found at http://youtu.be/s7NhrPqaVXg.

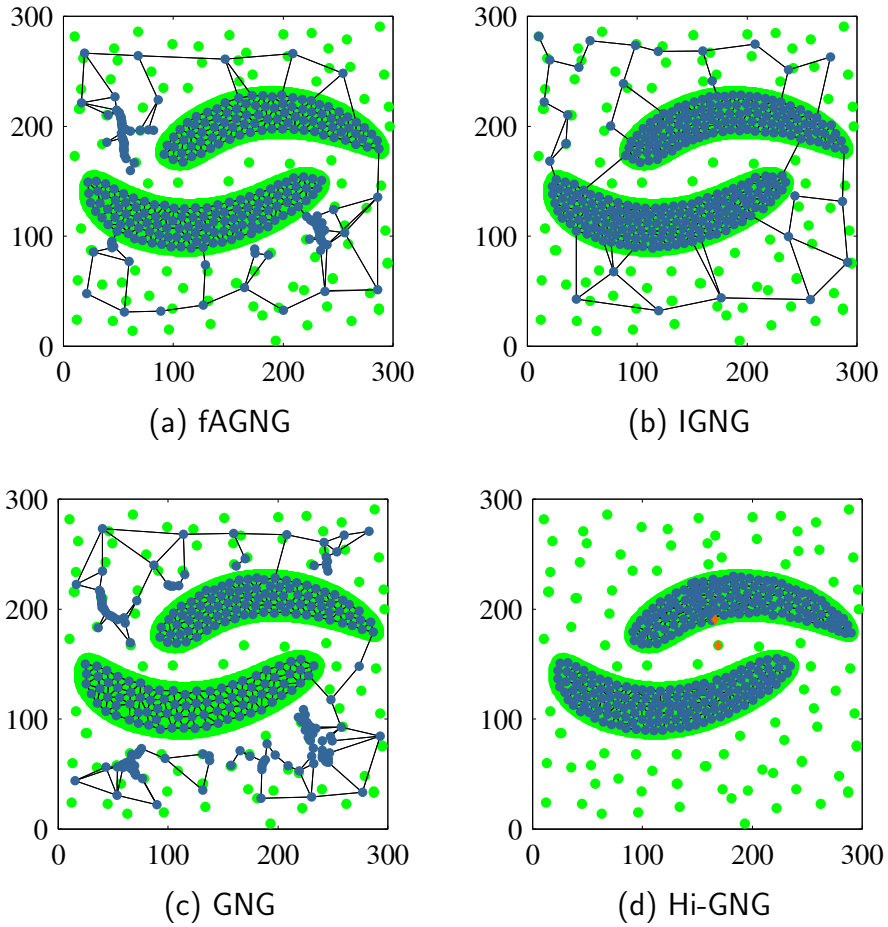


Figure 6-7: The comparable results for dataset 2 when the number of iterations was 500,000. The maximum number of neurons was limited at 250 for GNG, fAGNG and IGNG. The parameters of GNG were: $\lambda = 250$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$, $\alpha = 0.5$, $d = 0.0005$. The parameters of fAGNG were as follows: $k = 3$, $\lambda = 250$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$, $\alpha = 0.5$, $d = 0.0005$. The parameters of IGNG were: $\alpha_{mature} = 20$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$. The parameters of Hi-GNG were as follows: $\sigma = 10$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\hat{m} = 30$, $\alpha = 50$.

In addition, we compared the result of Hi-GNG with those of fAGNG, IGNG and GNG. In this experiment, the maximum number of neurons in GNG, fAGNG and IGNG was set to 250. We captured the experimental snapshots when the number of iterations was 500,000, because all the algorithms became stable at that time (Fig. 6-7). We observed that fAGNG and IGNG placed some neurons in the low-density areas and all the neurons in the network were connected with each other, which means that there was only one component in the graph, indicating that only

one cluster could be reported (shown in Fig. 6-7(a) and Fig. 6-7(b), respectively). The GNG algorithm reported two components in its topological graph, indicating that there were two clusters in the given dataset (Fig. 6-7(c)). However, the two clusters found by GNG were not the true clusters. On the other hand, the Hi-GNG algorithm automatically inserted a suitable number of neurons to generate the topological structure without being provided the maximum number of neurons in the network. The number of mature neurons became stable at around 270 and the number of immature neurons fluctuated around six. Furthermore, the generated topological structure always clearly reported that there were two clusters which were exactly the true clusters in the given dataset (Fig. 6-7(d)).

6.4.2 Incremental Learning

In the following experiments, we evaluated the performance of Hi-GNG for incremental learning and compared the experimental results with that of the GNG algorithm.

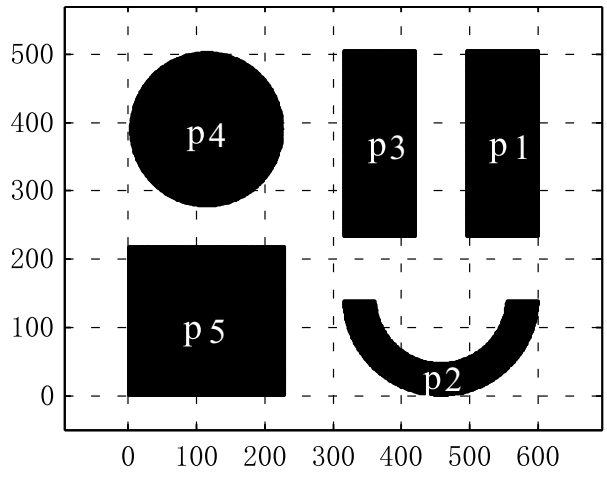


Figure 6-8: The synthetic dataset used in the experiment of incremental learning. The marker on the clusters indicates the order in the procedure of the data generation, e.g. "p4" indicates that the cluster of data is the 4^{th} part in the data generation.

We used a synthetic dataset, containing five parts (also the clusters) with 163,043 data points in total (Fig. 6-8). To test the performance of incremental learning based on learned structures, first, we input "p1" as the part of data that the algorithms had

already learned, and then, we sequentially input the remaining parts of data with an order as "p2, p3, ..., p5" after every 35,000 iterations. In the learning procedure of each part of data, we fairly generate data, i.e. the probabilities of generating data from the newly added part and from the learned parts were the same. Since the number of neurons in the GNG network would continuously increase, we limited its maximal number to 650. The learning quality was evaluated by MSE (Equation (2.2).

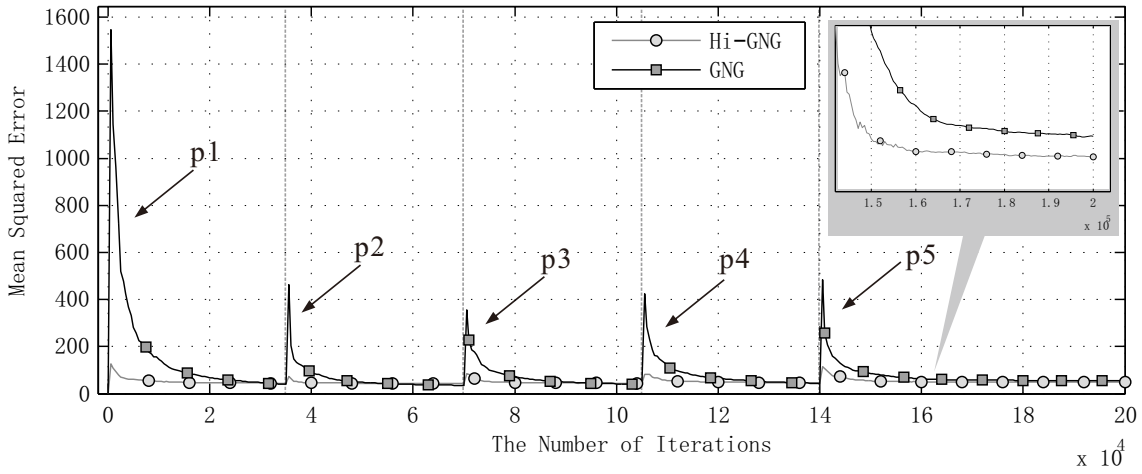


Figure 6-9: The learning qualities when each part of data arrivals with an order "p1, p2, ..., p5".

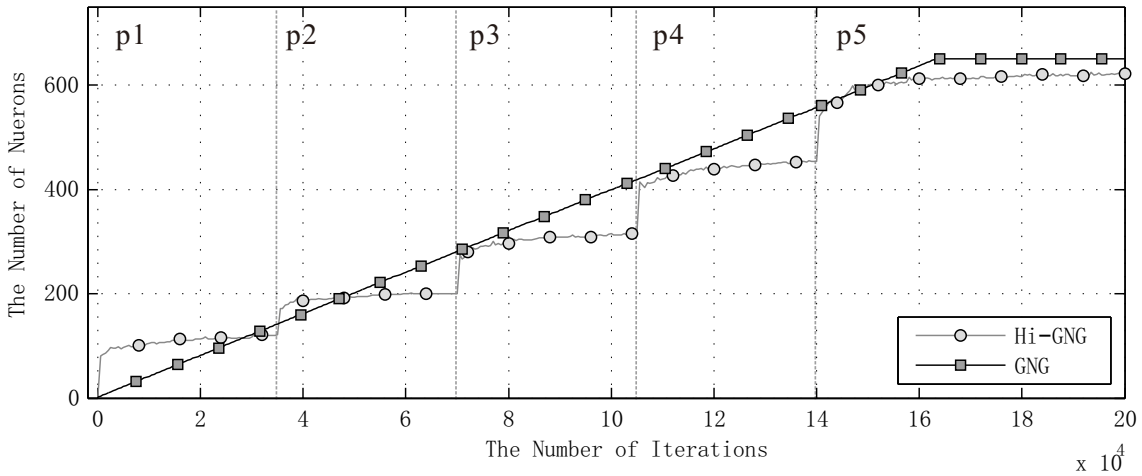


Figure 6-10: The number of neurons when each cluster of data arrivals with an order "p1, p2, ..., p5".

The results of the experiment regarding the learning quality are shown in Figure 6-9. First of all, for the learning procedure of each part of data, we observed that Hi-GNG could converge to a minimal MSE value faster than GNG. Second, for the first three parts of data after learning "p1", i.e. for "p2", "p3" and "p4", Hi-GNG and GNG converged at a very similar MSE value. However, for the last part, i.e. "p5", GNG could not converge at a minimized MSE value as good as it obtained before. The main reason was that the number of neurons in the GNG network reached to the predefined maximal value, i.e. 650, when it was learning the fifth part of data (Fig. 6-10). Hence, the number of neurons used to represent the last part of data was too small, which led to a bad quantization result. Although we could use a larger value for the predefined maximal number of neurons, it is easy to image that the number of neurons would definitely reach such an upper-limitation, when the amount of data increases to a considerable large value. Another possible solution was to use a larger value for the parameter λ, so that GNG could generate neurons with a longer time interval, but the cost was a slower adaption for the learning of each part of data. In addition, we also noticed that the number of neurons in the GNG network was linearly increasing until it reached the predefined maximal value, while on the other hand, the number of neurons in the Hi-GNG network increased like climbing stairs during the learning procedure of each part of data (Fig. 6-10). The number

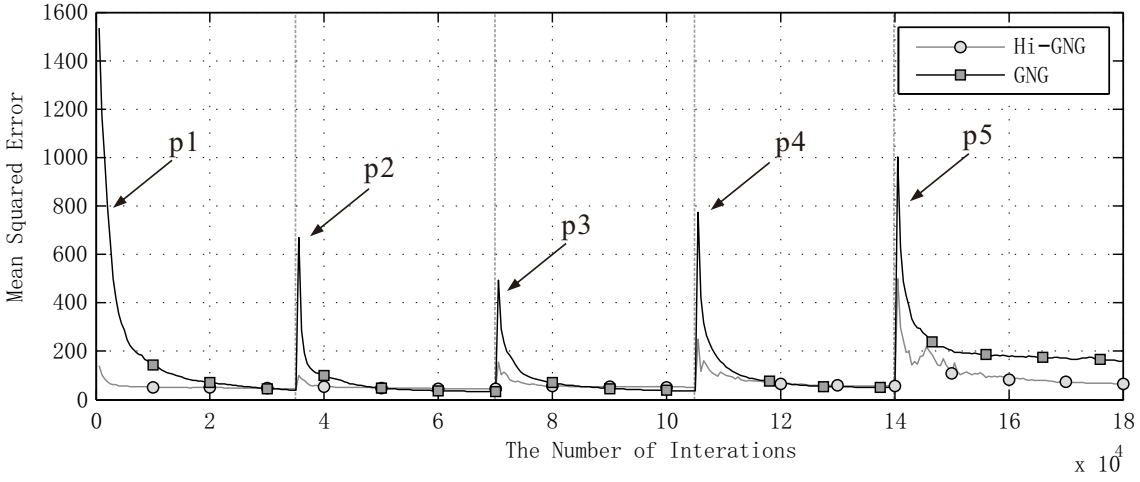
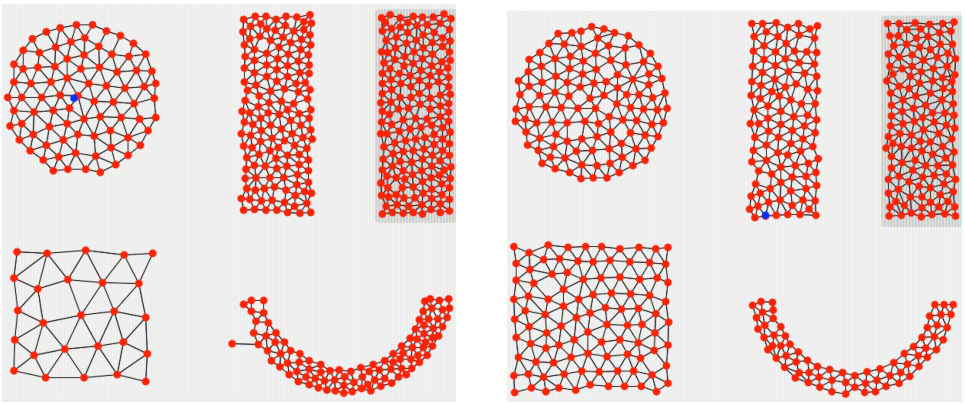


Figure 6-11: The learning qualities when the number of neurons was 500.

became stable at around 635 which was smaller than that of GNG, and meanwhile, quantization quality was better than that of the later one. This result was easier to be observed when we limited the maximal number of neurons in GNG network to 500 (Fig. 6-11), because less neurons were used to approximate the distribution of the last part of data, resulting in a larger MSE value. The snapshots when GNG and Hi-GNG finished learning the whole dataset within 500 neurons are shown in Fig. 6-12(a) and Fig. 6-12(b), respectively, where we could see that the Hi-GNG algorithms obtained a better presentation for the input data distribution than GNG.



(a) GNG (b) Hi-GNG

Figure 6-12: The snapshots when GNG (a) and Hi-GNG (b) finished learning the whole dataset. The parameters of GNG were: $\lambda = 250$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\alpha_{max} = 80$, $\alpha = 0.5$, $d = 0.0005$. The parameters of Hi-GNG were as follows: $\sigma = 24$, $\epsilon_b = 0.05$, $\epsilon_n = 0.006$, $\hat{m} = 30$, $\alpha = 80$.

6.4.3 An Efficiency Comparison between Improved Landmark FN-DBSCAN and Hi-GNG

In the following experiments, we focused on the efficiency comparisons between the two methods proposed in this dissertation, i.e. the improved landmark FN-DBSCAN algorithm (Chapter 5) and the Hi-GNG algorithm (Chapter 6). We used an large synthetic dataset, the volume of which will gradually increase by being extended with new batches of data points. Here, each batch of data points was an entire dataset used in Section 6.4.2 (shown in Fig. 6-8). In this experiments, we generated ten

batches of data points, and each data batch contained 163,043 data points belonging to five clusters. Hence, the final dataset contained 50 clusters including 1,630,430 data points in total. The procedure of data generation is illustrated in Fig. 6-13. In order to well learn each batch of data points, we carefully selected the parameters for the two algorithms, so that both of them could generate 60 landmarks or neurons for each new batch of data points. The experiments were conducted by Intel Core i7 3.40 GHz CPU with 8GB RAM running on Windows 7 64-bit operating system.

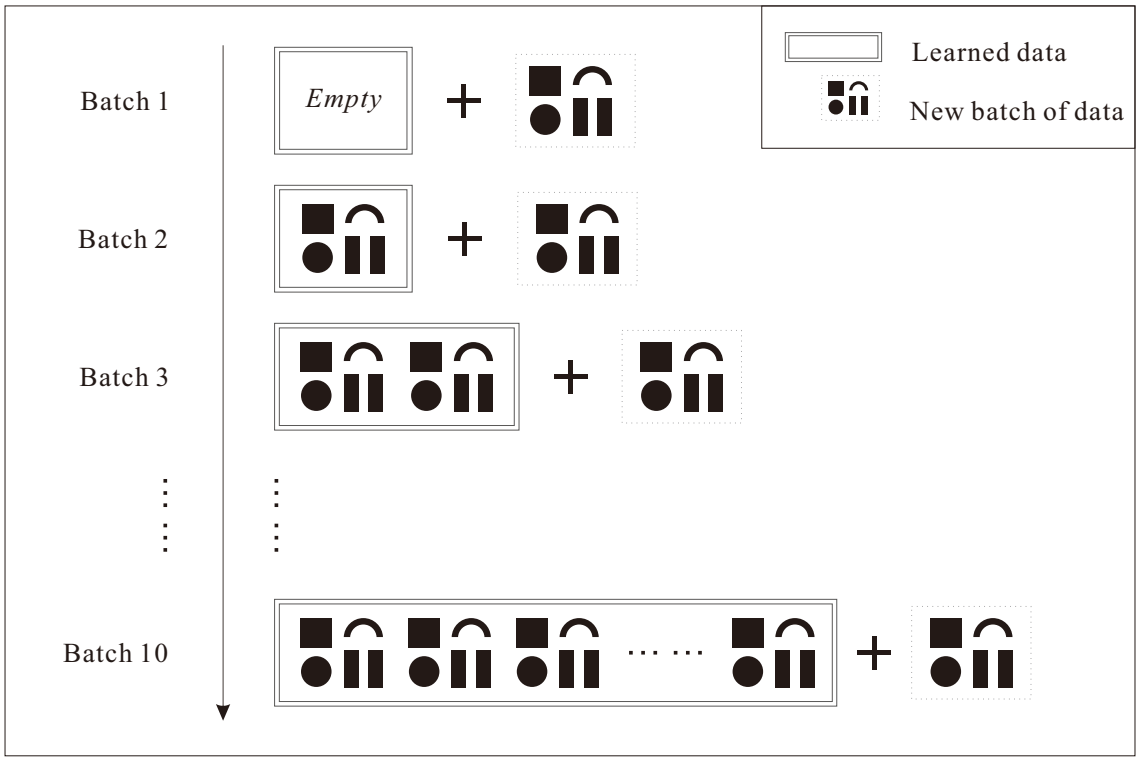


Figure 6-13: The procedure of data generation.

We show the efficiency comparisons in Fig. 6-14. We observed that, at the beginning, i.e. when the number of data was smaller than 326,086, the improved landmark FN-DBSCAN algorithm could run faster than Hi-GNG. However, with the increasing of the volume of the dataset, Hi-GNG significantly outperformed than its competitor, because it took an incremental clustering strategy and the improved landmark FN-DBSCAN algorithm had to calculate all the data points including the previously learned ones and the new ones.

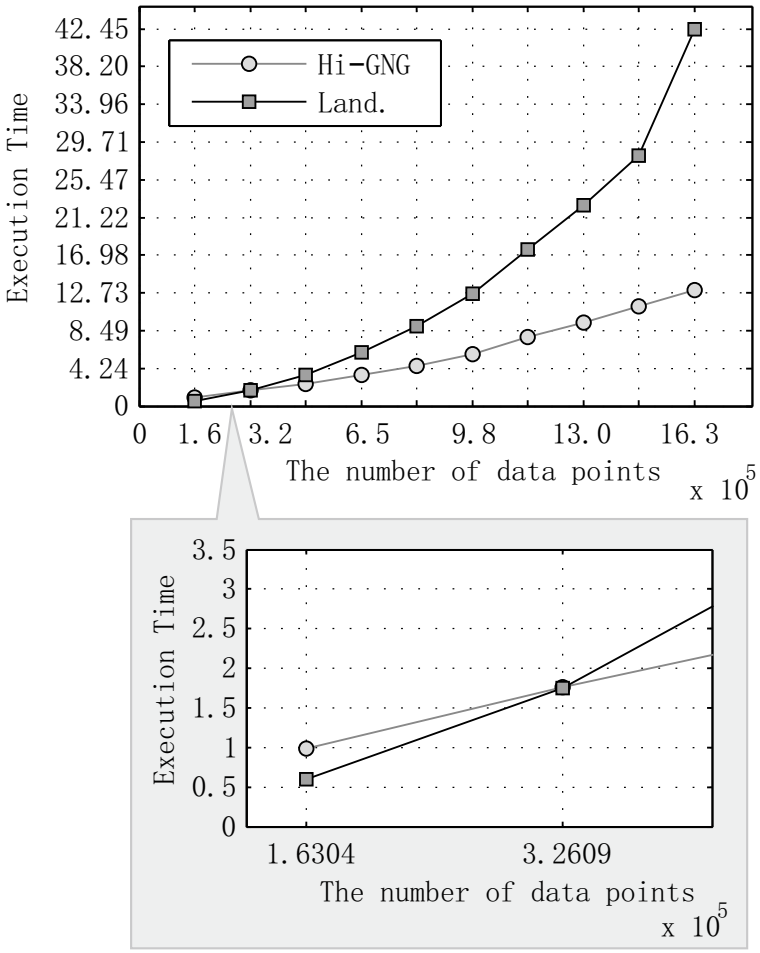


Figure 6-14: The efficiency comparison between the improved landmark FN-DBSCAN algorithm and the Hi-GNG algorithm.

6.5 Summary

In this chapter, we first proposed a new topology generation method, called enhanced competitive Hebbian learning (enhanced CHL), which considers how to both create and eliminate connections between neurons. On the basis of the enhanced CHL method, we further proposed a novel incremental self-organizing neural network, called enhanced incremental growing neural gas (Hi-GNG). The experiments presented in this chapter proved that the Hi-GNG algorithm could efficiently learn the given data distribution and the generated topological structure was not influenced by noisy data. In addition, Hi-GNG could automatically generate a topological structure

with a suitable number of neurons.

Although Hi-GNG is usually faster than the original GNG algorithm, the efficiency can be further improved by indexing the neurons by some supporting structures, such as R^*-tree, slim-tree or kd-tree, in order to reduce the cost related with the search for the nearest (or the second-nearest) neuron in the training procedure.

Some promising applications of the Hi-GNG algorithm are incremental clustering with noisy datasets, image segmentation and 3-D surface reconstruction.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, we have proposed two novel methods which are an efficient and robust density-based clustering and an incremental self-organizing neural network, respectively.

Since the original FN-DBSCAN algorithm has a time complexity of $O(n^2)$ which usually means an expensive cost on time. To improve the efficiency, in Chapter 3, we first presented our preliminary work, called landmark fuzzy neighborhood DBSCAN (landmark FN-DBSCAN). A new concept, called landmark, was introduced to represent a subset of the input dataset so that the original dataset can be compressed and represented by the generated landmarks. We gave a theoretical analysis on the time and space complexities of the algorithm, which showed that both of them were linear to the size of the dataset. The experiments presented in Chapter 3.3 showed that the landmark FN-DBSCAN algorithm was much more efficient than FN-DBSCAN, and also could provide a very similar quality of clustering to the original algorithm. However, there was an obvious limitation in this preliminary work, i.e. the input data distribution represented by the generated landmarks was influenced by the input data orders if the number of the generated landmarks was very small.

To improve this limitation, in Chapter 4, we proposed a new effective and efficient solution for the task of landmark generation, called Locally Adaptive Incremental

LBG (LAI-LBG). LAI-LBG could incrementally insert and remove landmarks with no requirement of predefining the number of landmarks. No matter how many landmarks were generated, it was always insensitive to the initializations and the input data orders, which means the input dataset could be well represented by the generated landmarks. In addition, the efficiency of the new algorithm was higher than its competitors.

In Chapter 5, on the basis of the previously proposed LAI-LBG and the landmark FN-DBSCAN algorithm, we further improved the previously proposed landmark FN-DBSCAN algorithm, which is one of the main contributions of this dissertation. The improved algorithm could obtain a good clustering result with a few number of landmarks, which indicates a further increased efficiency compared to the previous algorithm. In addition, the new algorithm was not sensitive to the input data orders, which means that the robustness was improved. Furthermore, the parameters used in the improved algorithm had typical values or could be automatically estimated, which means that the practicability was also enhanced.

Finally, as another main contribution of this dissertation, in Chapter 6, we proposed a novel self-organizing neural network for clustering called enHanced Incremental Growing Neural Gas (Hi-GNG) based on a new topology generation method which was an enhancement of competitive Hebbian learning (enhanced CHL). In this work, we modeled the birth and death of neurons, the creation and elimination of connections between each pair of neurons, and especially the adjustment of the connection-strength of each connection. Hence, the Hi-GNG algorithm presented a natural way to the incremental clustering techniques with its dynamic network structure. In addition, it could automatically and efficiently generate a topological structure with a suitable number of neurons, and was also robust to noisy data.

As a summary, we show the relationship between the proposed methods and the state of the art in Figure 7-1.

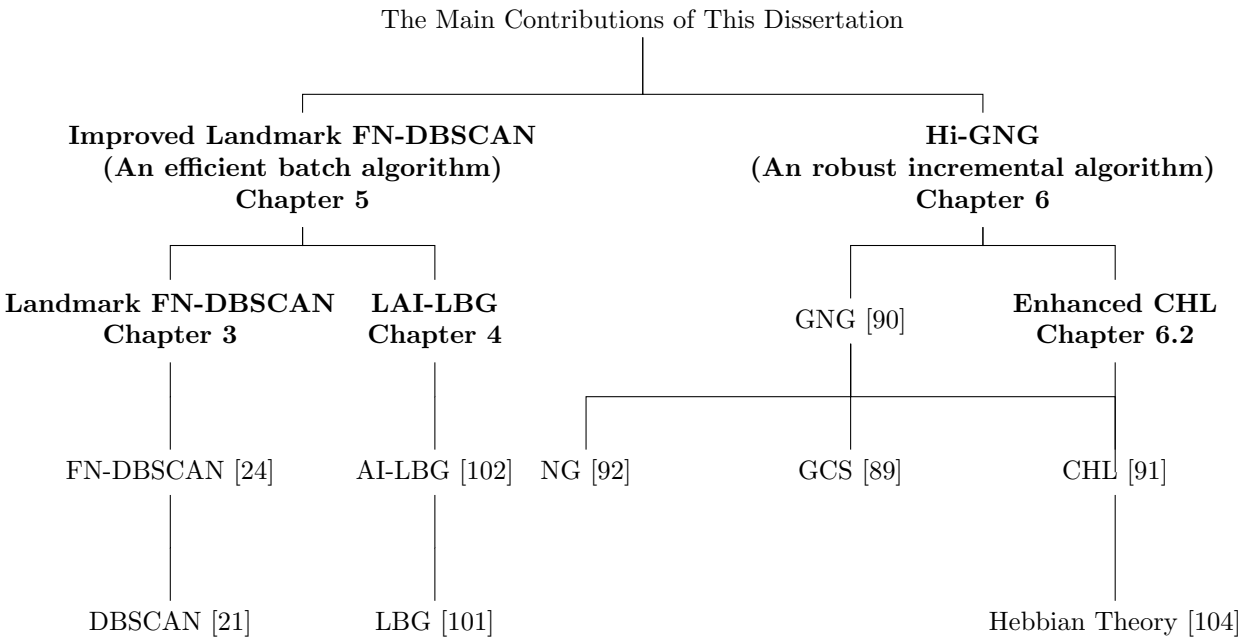


Figure 7-1: The relationship between the proposed methods and the state of the art.

7.2 Future work

We are considering about several extensions of the methods which were proposed in this dissertation as follows.

Like many density-based clustering algorithms, it is very hard for the landmark FN-DBSCAN algorithm and its improved version to detect overlapped clusters. However, it is very common that a real-world dataset contains such kind of clusters. Pioneers studied this task in many aspects, such as graph-based method [105, 106, 107], model-based method [108] and density constraint-based methods [109, 84]. It will make the proposed method more robust to real-world datasets, if we can bring a new feature of detecting overlapped clusters to them.

Although the landmark FN-DBSCAN and its improvement are presented as density-based clustering algorithms in this dissertation, it is passible that we can generalize them as a general efficient clustering framework which is a three-phase method described as follows.

1. Use LAI-LBG to generate landmarks so that the input dataset can be well

compressed into a smaller number of landmarks.

2. Use a specific clustering algorithm, e.g. a hierarchical clustering algorithm or a density-based clustering algorithm, to classify the generated landmarks into clusters.

3. Label data according to the clustering information about landmarks.

Let n and k be the number of data points and the number of the generated landmarks, respectively. In the second phase, we believe that it is possible to use an algorithm which has a high time-complexity, e.g. $O(k^2)$ or $O(k^3)$, because a high time complexity in the second phase cannot result in a big time consumption to the whole algorithm, if the condition $k \ll n$ is satisfied. Hence, it will be a promising work for us to create such a framework in order to improve the efficiency of the algorithms which have a high time complexity. Here we must note that, unlike some hybrid clustering, e.g. CSM [10] and CHAMELEON [19] which are the combination of k-means clustering and hierarchical clustering, the framework is not under the influence of initializations, because from Chapter 4 we know that the LAI-LBG is insensitive to the initializations.

The proposed LAI-LBG algorithm can work well as a landmark generation method, but we realized that it also can serve as a general vector quantization method. More experiments, such as the compression and decompression test on benchmark images, will be designed and conducted regarding this topic.

A limitation of the Hi-GNG algorithm is that the parameter, σ, may influence the final result. Unfortunately, so far it is a very difficult task to determine a suitable value for it, because the data are incrementally input to the algorithm. In order to improve the practicability, our further work will also include a method which can automatically determine the parameter, σ.

Appendix A

Related Methods Involved in this Dissertation

A.1 Outliers Detection

Given a dataset, an outlier is a data point located significantly distant from the rest of the data points. It is very common that a given dataset contains some outliers. Sometimes, an outlier may indicate an exceptional data point and generally should be labeled in a robust clustering algorithm. In the statistics field, several outliers detecting methods have been proposed, such as Grubbs' test [110], Tiejen-Moore test [111] or the generalized extreme studentized deviate (GESD) test [112]. Here, we briefly introduce the Grubbs' test which is an effective statistical way of outliers detection in a univariate dataset.

Let the sample of n observations be denoted in order of increasing magnitude by $x_1 \leq x_2 \leq x_3 \ ... \leq x_n$ and let x_n be the doubtful value, i.e. the largest value. The following test criterion, T_n, is used to detect one single outlier.

$$T_n = (x_n - \bar{x})/s \tag{A.1}$$

where $\bar{x}$ and s are the arithmetic average and the standard deviation of all n values, respectively.

Then x_n is considered as an outlier if the following condition is met.

$$T_n > \frac{n-1}{\sqrt{n}} \sqrt{\frac{(t_{\alpha/(n),n-2})^2}{n-2+(t_{\alpha/(n),n-2})^2}} \tag{A.2}$$

where α is the *significance level* and $t_{\alpha/(n),n-2}$ denotes the upper critical value of the student's t-distribution with $n-2$ degrees of freedom.

The Grubbs' test can also be used to detect the outlier which is the smallest one of n observations. Interested readers are recommended to read the literature[110] for more detailed discussions. In this dissertation, we only use Grubb's test to detect the potential outlier which is the largest one of n observations.

A.2 Data Normalization

Given a dataset X as an n-by-m matrix, where n is the number of data points and m is the dimension of data points, considering a data point as a row-vector, then x_{ik}, which is the k^{th} (k=1,2,...,m) vector component of the data point x_i, is transformed to:

$$x_{ik}{}' = \frac{x_{ik} - x_k^{\min}}{(x_k^{\max} - x_k^{\min}) * \sqrt{m}} \tag{A.3}$$

where $x_k^{\min} = \min\{X(:,k)\}$ and $x_k^{\max} = \max\{X(:,k)\}$ where $X(:,k)$ is the k^{th} column of X. $1/\sqrt{m}$ indicates that the maximum distance between the normalized data points belongs to the range of [0, 1]. See more detailed discussions in the literature [24].

A.3 Rand-Index

Let $\boldsymbol{C} = \{C_1, C_2, ..., C_I, ..., C_K\}$ be a set of clusters obtained by a clustering method, where C_I is the I^{th} cluster of data points, K is the total number of clusters. Similarly, let $\boldsymbol{P}$ be the true partitions which are described as $\boldsymbol{P} = \{P_1, P_2, ..., P_J, ..., P_L\}$, where P_J is the J^{th} partition in the true partitions, L is the total number of partitions. Suppose that there is a pair of data points (x_v, x_u) in the dataset,

then it is referred as follows: (SS): If both data points belong to the same cluster $\boldsymbol{C}$ and they are also in the same partition in the true partition $\boldsymbol{P}$. SS $= \{(x_v, x_u) \mid \exists I(x_v \in C_I \wedge x_u \in C_I)\ and\ \exists J(x_v \in P_J \wedge x_u \in P_J)\}$. (DD): if two data points belong to different clusters of $\boldsymbol{C}$ and also belong to different partitions of P. DD $= \{(x_v, x_u) \mid \nexists I(x_v \in C_I \wedge x_u \in C_I)\ and\ \nexists J(x_v \in P_J \wedge x_u \in P_J)\}$. Assuming that a, b are the numbers of SS, DD respectively and M denotes the total number of such pairs of data points, which is $\binom{n}{2}$, the Rand-Index, R, is given by Equation (A.4).

$$R = (a + b)\,/M \tag{A.4}$$

Here, $R \in [0, 1]$ represents the clustering quality provided by the tested method: the higher, the better.

Bibliography

[1] Roger Porkess. *A world full of data: Statistics opportunities across A level subjects*. Royal Statistical Society, 2013.

[2] Steve Lohr. The age of big data. *New York Times*, 11, 2012.

[3] Merriam-Webster Online Dictionary with the keyword "cluster analysis". http://www.merriam-webster.com/dictionary/, 2014.

[4] Xu Rui and Wansch II Donald. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[5] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[6] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[7] Huang Zhexue. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 304(3):283–304, 1999.

[8] Raymond T. Ng and Jiawei Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[9] Chris H. Q. Ding and Xiaofeng He. K-nearest-neighbor consistency in data clustering: incorporating local information into global optimization. In *Proceedings of the ACM symposium on Applied computing*, pages 584–589, 2004.

[10] Cheng-Ru Lin and Ming-Syan Chen. Combining partitional and hierarchical algorithms for robust and efficient data clustering with cohesion self-merging. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):145–159, 2005.

[11] Ming-Chao Chiang, Chun-Wei Tsai, and Chu-Sing Yang. A time-efficient pattern reduction algorithm for k-means clustering. *Information Sciences*, 181(4):716 – 731, 2011.

[12] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[13] K.-L. Du. Clustering: A neural network approach. *Neural Networks*, 23(1):89–107, 2010.

[14] Yuqing Song, Shuyuan Jin, and Jie Shen. A unique property of single-link distance and its application in data clustering. *Data & Knowledge Engineering*, 70(11):984–1003, 2011.

[15] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[16] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 4th edition, 2009.

[17] Robin Sibson. SLINK: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[18] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[19] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[20] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[22] E. N. Nasibov. A robust algorithm for solution of the fuzzy clustering problem on the basis of the fuzzy joint points method. *Cybernetics and Systems Analysis*, 44(1):7–17, 2008.

[23] Efendi N. Nasibov and Gözde Ulutagay. A new unsupervised approach for fuzzy clustering. *Fuzzy Sets and Systems*, 158(19):2118–2133, 2007.

[24] Efendi N. Nasibov and Gözde Ulutagay. Robustness of density-based clustering methods with various neighborhood relations. *Fuzzy Sets and Systems*, 160(24):3601 – 3615, 2009.

[25] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. In *Proceedings of ACM SIGMOD International Conference on Management of Data. S*, volume 197, pages 197–208, 1994.

[26] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 802–813, 2003.

[27] Michael Stonebraker, Jim Frew, Kenn Gardels, and Jeff Meredith. The sequoia 2000 storage benchmark. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 2–11. ACM, 1993.

[28] Victoria Woollaston. Revealed, what happens in just one minute on the internet: 216,000 photos posted, 278,000 tweets and 1.8m facebook likes. http://www.dailymail.co.uk/sciencetech/article-2381188/revealed-happens-just-one-minute-internet-216-000-photos-posted-278-000-tweets-1-8m-facebook-likes.html, 2013.

[29] Min Chen, Shiwen Mao, Yin Zhang, and Victor CM Leung. Big data analysis. In *Big Data*, pages 51–58. Springer, 2014.

[30] Mario AT Figueiredo and Anil K. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):381–396, 2002.

[31] Mark H Hansen and Bin Yu. Model selection and the principle of minimum description length. *Journal of the American Statistical Association*, 96(454):746–774, 2001.

[32] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

[33] Mu-chun Su and Yi-chun Liu. A new approach to clustering data with arbitrary shapes. *pattern recognition*, 38(11):1887–1901, 2005.

[34] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[35] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley and Sons, New York, 1990.

[36] Geoffrey H Ball and David J Hall. Isodata, a novel method of data analysis and pattern classification. Technical report, Stanford Research Institute, 1965.

[37] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, 2002.

[38] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281, 1999.

[39] Radha Chitta, Rong Jin, Timothy C. Havens, and Anil K. Jain. Approximate kernel k-means: Solution to large scale kernel clustering. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 895–903. ACM, 2011.

[40] Joseph C Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1973.

[41] James C Bezdek. *Pattern recognition with fuzzy objective function algorithms*. Kluwer Academic Publishers, 1981.

[42] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2):191–203, 1984.

[43] Timothy C Havens, James C Bezdek, Christopher Leckie, Lawrence O Hall, and Marimuthu Palaniswami. Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146, 2012.

[44] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *in proceedings of 15th International Conference on Data Engineering*, pages 512–521, 1999.

[45] Ying Zhao, George Karypis, and Usama Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.

[46] Benjamin CM Fung, Ke Wang, and Martin Ester. Hierarchical document clustering using frequent itemsets. In *Proceedings of the Third SIAM International Conference on Data Mining*, volume 3, pages 59–70, 2003.

[47] Alexander Kraskov and Peter Grassberger. Mic: mutual information based hierarchical clustering. In *Information theory and statistical learning*, pages 101–123. Springer, 2009.

[48] Clark F Olson. Parallel algorithms for hierarchical clustering. *Parallel computing*, 21(8):1313–1325, 1995.

[49] Elias Dahlhaus. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2):205–240, 2000.

[50] Sanguthevar Rajasekaran. Efficient parallel hierarchical clustering algorithms. *IEEE Trans. Parallel Distrib. Syst.*, 16(6):497–502, 2005.

[51] Katherine A Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. In *Proceedings of the 22nd international conference on Machine learning*, pages 297–304, 2005.

[52] Sanjoy Dasgupta and Philip M Long. Performance guarantees for hierarchical clustering. *Journal of Computer and System Sciences*, 70(4):555–569, 2005.

[53] Caiming Zhong, Duoqian Miao, and Pasi Fränti. Minimum spanning tree based split-and-merge: A hierarchical clustering method. *Information Sciences*, 181(16):3397–3410, 2011.

[54] David Cheng, Ravi Kannan, Santosh Vempala, and Grant Wang. A divide-and-merge methodology for clustering. In *ACM Transactions on Database Systems*, pages 196–205, 2005.

[55] Peter HA Sneath. The application of computers to taxonomy. *Journal of general microbiology*, 17(1):201–226, 1957.

[56] T. Sorensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyzes of the vegetation on danish commons. *Biologiske Skrifter*, 5:1–34, 1948.

[57] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[58] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[59] Fionn Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[60] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[61] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[62] Fionn Murtagh. Multidimensional clustering algorithms. *Compstat Lectures, Vienna: Physika Verlag*, 1, 1985.

[63] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1998.

[64] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data and Symposium on Principles of Database Systems*, pages 49–60, 1999.

[65] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Optics-of: Identifying local outliers. In *Principles of data mining and knowledge discovery*, pages 262–270. Springer, 1999.

[66] Elke Achtert, Christian Böhm, and Peer Kröger. Deli-clu: boosting robustness, completeness, usability, and efficiency of hierarchical clustering by a closest pair ranking. In *Advances in Knowledge Discovery and Data Mining*, pages 119–128. Springer, 2006.

[67] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, and Arthur Zimek. Finding hierarchies of subspace clusters. In *Knowledge Discovery in Databases: PKDD 2006*, pages 446–453. Springer, 2006.

[68] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[69] Alexander Hinneburg and Daniel A Keim. A general approach to clustering in large databases with noise. *Knowledge and Information Systems*, 5(4):387–415, 2003.

[70] Alexander Hinneburg and Hans-Henning Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *Advances in Intelligent Data Analysis VII*, pages 70–80. Springer, 2007.

[71] P. Viswanath and V. Suresh Babu. Rough-DBSCAN: A fast hybrid density based clustering method for large data sets. *Pattern Recognition Letters*, 30(16):1477–1488, 2009.

[72] Xiaowei Xu, Martin Ester, H-P Kriegel, and Jörg Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings of 14th International Conference on Data Engineering*, pages 324–331. IEEE, 1998.

[73] Chris Fraley and Adrian E. Raftery. Enhanced model-based clustering, density estimation, and discriminant analysis software: MCLUST. *Journal of Classification*, 20(2):263–286, 2003.

[74] B. Borah and D.K. Bhattacharyya. A clustering technique using density difference. In *Proceedings of the Signal Processing, Communications and Networking*, pages 585–588, 2007.

[75] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[76] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *Proceedings of 41st annual symposium onFoundations of computer science*, pages 359–366. IEEE, 2000.

[77] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment, 2003.

[78] Olfa Nasraoui and Carlos Rojas. Robust clustering for tracking noisy evolving data streams. In *Proceedings of SIAM International Conference on Data Mining*. SIAM.

[79] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. In *Neural Information Processing Systems (NIPS)*, pages 10–18, 2009.

[80] Marcel R Ackermann, Marcus Märtens, Christoph Raupach, Kamil Swierkot, Christiane Lammersen, and Christian Sohler. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17(1):2–4, 2012.

[81] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of Data*, pages 103–114. ACM, 1996.

[82] Samer Nassar, Jörg Sander, and Corrine Cheng. Incremental and effective data summarization for dynamic hierarchical clustering. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 467–478. ACM, 2004.

[83] Pedro Pereira Rodrigues, João Gama, and Joao Pedro Pedroso. Hierarchical clustering of time-series data streams. *IEEE Transactions on Knowledge and Data Engineering*, 20(5):615–627, 2008.

[84] Feng Cao, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of SIAM International Conference on Data Mining*. SIAM, 2006.

[85] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.

[86] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.

[87] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, 2001.

[88] Paul Mangiameli, Shaw K. Chen, and David West. A comparison of som neural network and hierarchical clustering methods. *European Journal of Operational Research*, 93(3):402–417, September 1996.

[89] Bernd Fritzke. Growing cell structures-a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7:1441–1460, 1994.

[90] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, volume 7, pages 625–632. MIT Press, 1995.

[91] T. M. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *The 1993 International Conference on Artificial Neural Networks (ICANN'93)*, pages 427–434, 1993.

[92] Thomas M. Martinetz, Stanislav G. Berkovitsch, and Klaus J. Schulten. "Neural-gas" network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4:558–569, 1993.

[93] Alexander Ocsa, Carlos Bedregal, and Ernesto Cuadros-Vargas. DB-GNG: A constructive self-organizing map based on densilty. In *The 2007 International Joint Conference on Neural Networks(IJCNN 2007)*, pages 1953–1958, 2007.

[94] Daniel Fišer, Jan Faigl, and Miroslav Kulich. Growing neural gas efficiently. *Neurocomputing*, 104:72 – 82, 2013.

[95] J Garcia-Rodriguez, Anastassia Angelopoulou, JM García-Chamizo, A Psarrou, S Orts-Escolano, and V Morell-Gimenez. Fast autonomous growing neural gas. In *The 2011 International Joint Conference on Neural Networks(IJCNN 2011)*, pages 725–732, 2011.

[96] Yann Prudent and Abdellatif Ennaji. An incremental growing neural gas learns topologies. In *The 2005 International Joint Conference on Neural Networks(IJCNN 2005)*, volume 2, pages 1211–1216, 2005.

[97] Shen Furao and Osamu Hasegawa. An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, 19(1):90–106, 2006.

[98] A.K. Qin and P.N. Suganthan. Robust growing neural gas algorithm with application in cluster analysis. *Neural Networks*, 17(8-9):1135 – 1148, 2004.

[99] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity methods: part i. *ACM Sigmod Record*, 31(2):40–45, 2002.

[100] Stuart Lloyd. Least squares quantization in PCM. *Bell Telephone Laboratories Memorandum 1957, also published in IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[101] Yoseph Linde, Andres Buzo, and Robert Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[102] F Shen and Osamu Hasegawa. An adaptive incremental LBG for vector quantization. *Neural Networks*, 19(5):694–704, 2006.

[103] Giuseppe Patané and Marco Russo. The enhanced LBG algorithm. *Neural Networks*, 14(9):1219–1237, 2001.

[104] D. O. Hebb. *The organization of behavior: a neuropsychological theory*. John Wiley & Sons, New York, 1949.

[105] Airel Pérez Suárez, José Fco Martínez Trinidad, Jesús A Carrasco Ochoa, and José E Medina Pagola. A new incremental algorithm for overlapped clustering. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 497–504. Springer, 2009.

[106] Xiaodan Yan, Stephen Kelley, Mark Goldberg, and Bharat B Biswal. Detecting overlapped functional clusters in resting state fmri with connected iterative scan: a graph theory based clustering algorithm. *Journal of neuroscience methods*, 199(1):108–118, 2011.

[107] Thiago Christiano Silva and Liang Zhao. Uncovering overlapping cluster structures via stochastic competitive learning. *Information Sciences*, 247:40–61, 2013.

[108] Arindam Banerjee, Chase Krumpelman, Joydeep Ghosh, Sugato Basu, and Raymond J Mooney. Model-based overlapping clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 532–537, 2005.

[109] Carlos Ruiz, Myra Spiliopoulou, and Ernestina Menasalvas. C-dbscan: Density-based clustering with constraints. In *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, pages 216–223. Springer, 2007.

[110] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[111] Gary L Tietjen and Roger H Moore. Some Grubbs-type statistics for the detection of several outliers. *Technometrics*, 14(3):583–597, 1972.

[112] Bernard Rosner. Percentage points for a generalized ESD many-outlier procedure. *Technometrics*, 25(2):165–172, 1983.