Scalable Model-based Clustering Algorithms for Large Databases and Their Applications

BY
HUIDONG JIN

SUPERVISED BY:
PROF. KWONG-SAK LEUNG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE AND ENGINEERING
©THE CHINESE UNIVERSITY OF HONG KONG
AUGUST, 2002

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the materials in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.
Scalable Model-based Clustering Algorithms for Large Databases and Their Applications

Submitted by
Huidong JIN

for the degree of Doctor of Philosophy
at The Chinese University of Hong Kong in August, 2002

Abstract

With the unabated growth of data amassed from business, scientific and engineering disciplines, cluster analysis and other data mining functionalities, play a more and more important role. They can reveal previously unknown and potentially useful patterns and relations in large databases. One of the most significant challenges in data mining is scalability — effectively handling large databases with linear computational complexity and limited main memory.

This thesis addresses the scalability problem of the mixture model-based clustering algorithms for databases with large number of data items. It proposes a scalable model-based clustering framework. The basic idea is first to partition a large data set into subclusters roughly, and then to generate clusters from summary statistics of these subclusters effectively. The procedure may be manipulated by a controller. The framework consists of three modules: data summarization, in-memory clustering, and optional controller.

The data summarization module sums up data sets by only storing appropriate summary statistics. Besides the BIRCH’s data summarization procedure, there exist many approaches. Our adaptive grid-based data summarization procedure simply partitions data space and sum up the data items within the same cells into summary statistics. Based on the principle of Self-Organizing Map (SOM), we
establish an expanding SOM and an integrated SOM for data summarization and projection. The two SOMs can generate better mappings than the traditional SOM in terms of both the quantization and the topological errors. It is also substantiated by the experiments where they can generate most accurate solutions of the travelling salesman problem so far in the neural network literature.

The in-memory clustering module generates mixture models from summary statistics of subclusters. We establish two model-based clustering algorithms based on the general Expectation-Maximization (EM) algorithm. If attributes are statistically independent, we use a clustering feature to represent a subcluster, and employ EMACF to generate clusters. Otherwise, our EMADS handles data summaries where the correlations are embedded. We prove the convergence of both algorithms. Combining with the grid-based data summarization procedure, we have two scalable clustering systems: the gEMACF and gEMADS algorithms. Similarly, the bEMACF and bEMADS algorithms work on the subclusters generated by the BIRCH’s data summarization procedure. They can run one or two orders of magnitude faster than the classical model-based clustering algorithm and generate results with no or little loss of accuracy. Their effectiveness and efficiency are confirmed by experiments on both synthetic and real world data.

The last optional module provides a controller to our clustering system. We introduce genetic algorithms to guide the model-based clustering techniques and establish the GAXEM algorithm. It determines the optimal number of clusters automatically. Some preliminary experimental results substantiate that the GAXEM algorithm performs well on both synthetic and real world data sets.

Besides the clustering application of our model-based techniques, we also briefly discuss two other applications.
论文摘要

对大型数据库可扩展的基于模型的聚类算法和应用
金慧东

随着商业、科学和工程领域中持续不断数据积累，聚类分析和其他数据挖掘功能一起，发挥着越来越重要的角色。数据挖掘能够在大数据库中揭示不清楚，又有潜在用途的模式和关系。可扩充性 — 用有限的内存有效地、线性地处理大数据库 — 是数据挖掘中最重要的挑战之一。

本论文旨在解决对大量数据项的数据集进行基于混合模型的聚类分析的可扩展性问题。它提出了一个可扩展的基于模型的聚类分析的框架。其基本思路是，首先将一个大型的数据集粗略的分成一些子集，然后从子集的概要统计中产生精确的聚类模型，这个过程还有个控制器进行调节。这框架有三个模块：数据汇总、驻内存的聚类分析、及可选的控制器。

数据汇总模块汇总整个数据集，并存到合适的概要统计中。除了 BIRCH 数据汇总程序，还存在很多其他方法。我们的自适应的、基于网格的数据汇总程序简单地划分数据空间，并把处于同一单元的数据项汇总到一个概要统计中。用自组织映射原理，我们建立了可用于数据汇总和投影的一个膨胀的自组织映射网和一个集成的自组织映射网。相比于传统的自组织映射网，这两个自组织映射网能够产生更小矢量化错误，更小拓扑错误的映射。他们能够产生在神经网络文献中最精确的旅行商问题的解也佐证了这一点。

驻内存的聚类分析模块从子集的概要统计中产生精确混合模型。由一般期望值最大化算法，我们建立了两个基于模型的聚类算法。如果数据属性相互独立，子集有聚类特征描述，EMACF 能够生成簇。另外，EMADS 能够处理含有相关性信息的数据概要。我们证明两个算法收敛。结合自适应的、基于网格的数据汇总程序，我们能建立两个可扩展的聚类分析系统：gEMACF 和 gEMADS 算法。类似地，结合 BIRCH 数据汇总程序，我们有 bEMACF 和 bEMADS 两个可扩展的聚类分析系统。他们能够比经典的基于模型聚类算法快一到二个数量级，并且在精确度上没有或者很少损失。他们的有效性和高效率在人造和现实世界数据集上的全面实验中得到了验证。

可选的控制器可调节我们的聚类分析过程。我们引入遗传算法来指导基于模型的聚类分析技术，建立了 GAXEM 算法。它能够自动生成最优簇个数。一些初步的实验证实 GAXEM 算法在人造和现实世界数据集上表现得相当好。我们也对如何结合 GAXEM 和 EMACF/EMADS 组成一些新的可扩展聚类算法进行了讨论。

除了将基于模型的技术应用于聚类分析，我们也对在孤立点检测和数据库查询优化上的应用进行了简短地讨论。
Acknowledgements

Throughout these four years, there were hurdles and difficulties, I have been lucky that helping hands were always around.

First, I must thank my thesis supervisor, Prof. Kwong-Sak Leung. He not only gives me kind guidance, encouragement and inspiration on my research, but also shows me much wisdom about life. Even though he has a very busy schedule, he is always available for help and advice. Most importantly, he has given me a free hand to pursue what appeals to me greatly. Without his open-mindedness, this thesis could not have been here.

I am in deep gratitude to Dr. Man-Leung Wong from Lingnan University, Hong Kong. He has been a constant source of advice and support for me from programming, article composition, proposal writing, thesis matters, to career planning. He always gives me instructive and prompt advice.

I must express my appreciation to Prof. Irwin K.C. King and Prof. Kin Hong Wong, and Prof. Sam T.W. Kwong. They serve in my thesis committee and have been giving me valuable suggestions on my research for many years.

I also acknowledge the great help from Prof. Zongben Xu. He first led me to the door of research with enthusiasm. I still enjoy the constructive and sound discussion with him on both research and life.

I thank the kind help from Wing-Ho Shum, who has implemented the Expanding Self-Organization Map (ESOM) for data summarization and visualization.

Many other friends and colleagues of mine have provided me great support throughout the years, and the following is an incomplete list. In CUHK: Prof.

I am indebted to my parents, Yonggui Jin and Guihua Qu, and my brother and sister, Huiqiang and Huiping, for they have given me a family filled with love and care.

Finally, my wife, Haiyan, deserves greater thanks than I can possibly give. Her supports for me in various forms have been enormous. She encouraged me to keep on finishing my thesis when I was in doubt. She did not complain that I stayed in office seven days a week and I have never accompanied her for a whole weekend. She has proofread my articles carefully. Without her love and patience, I could never have come so far. I appreciate you, my love.
## Contents

Abstract | i
--- | ---
Acknowledgements | iv

### 1 Introduction

1.1 Data mining | 1
1.2 Cluster Analysis | 3
  1.2.1 Challenges of Cluster Analysis in Data Mining | 4
  1.2.2 A Survey of Cluster Analysis in Data Mining | 6
1.3 The Thesis | 11
  1.3.1 Motivation | 11
  1.3.2 Thesis Statement | 12
  1.3.3 Contributions of the Thesis | 12
1.4 Thesis Roadmap | 14

### 2 Model-based Cluster Analysis

2.1 Mixture Model and Clustering | 16
  2.1.1 Mixture Model | 16
  2.1.2 Likelihood | 18
2.2 The General EM Algorithm | 19
2.3 The EM Algorithm | 20
2.4 Hierarchical Agglomerative Clustering | 22
2.5 Model Selection ................................................. 22
2.6 Chapter Summary ............................................. 24

3 Scalable Model-based Clustering Framework ........................................ 25
  3.1 Scalable Clustering Techniques ..................................... 25
  3.2 Multi-phase Model-based Clustering Framework ..................... 28
    3.2.1 Data Summarization ..................................... 29
    3.2.2 In-memory Model-based Clustering ......................... 33
    3.2.3 Optional Controller ..................................... 33
  3.3 Features ..................................................... 34
  3.4 Chapter Summary ............................................. 35

4 Data Summarization Procedures ............................................. 36
  4.1 Adaptive Grid-based Data Summarization ......................... 36
  4.2 SOM for Data Summarization .................................. 42
    4.2.1 Self-Organization Map .................................. 42
    4.2.2 Expanding SOM ........................................ 46
    4.2.3 Integrated SOM ........................................ 59
  4.3 Data Summarization Procedure of BIRCH ......................... 61

5 Scalable Cluster Analysis by Working on Clustering Features ............ 64
  5.1 EMACF ...................................................... 64
    5.1.1 Motivation ............................................. 64
    5.1.2 Algorithm Derivation ................................... 66
    5.1.3 Convergence Guarantee .................................. 71
    5.1.4 Complexity ............................................ 73
  5.2 Performance of gEMACF ....................................... 73
    5.2.1 Methodology ............................................ 74
    5.2.2 Evaluation Metrics and Data Generation .................... 75
    5.2.3 Clustering Accuracy .................................... 79
## List of Tables

4.1 Average quantization errors of ESOM and SOM. .................................. 57
4.2 Average topological errors of ESOM and SOM. ................................. 58
4.3 Average execution times of ESOM and SOM. ................................. 59

5.1 The clustering accuracy of five clustering algorithms on 10 data sets. ......................................................... 79
5.2 The clustering accuracy on eight 4-dimensional data sets. ........... 82
5.3 The clustering accuracy of 5 clustering algorithms on 10 data sets. 87
5.4 Clustering accuracy of five clustering algorithms for eight 4-dimensional data sets. ......................................................... 91
5.5 The performance of four clustering algorithms on the Forest Cover-Type Data. ......................................................... 93
5.6 The performance of four clustering algorithms on the Census-Income Database. ......................................................... 94
5.7 The performance of four clustering algorithms on the California housing data. ......................................................... 95

6.1 The clustering accuracy of the four clustering algorithms on the seven synthetic data sets. $N, D, K$ and $M$ indicate the number of data items, the data dimensionality, the number of clusters, and the number of subclusters, respectively. .................................. 117
6.2 The clustering accuracy of the four clustering algorithm on eight 4-dimensional data sets. ......................................................... 120
6.3 The performance of the bEMADS algorithm on the Forest Cover-Type Data. ........................................ 121
6.4 The performance of the bEMADS algorithm on the Census-Income Database. ........................... 122
6.5 The performance of bEMADS on the California housing data. .................................................. 123
6.6 The computation complexity and the memory requirements of the in-memory clustering algorithms. $N$, $D$, $K$, $I$, and $M$ indicate the total number of data items, the data dimensionality, the number of clusters, the number of iterations, and the number of subclusters, respectively. .......................................... 125

7.1 The experimental results on nine synthetic data of EnumEM, AutoClass, GAEM, and GAXEM. $K$ and $N$ indicate respectively the optimal number of clusters and the number of data items in a data set. ‘Accu’(%) is the average classification accuracy value and ‘Suc’ indicates the number of successful runs within 10 runs. The time unit is seconds. ......................................................... 138
7.2 The experimental results of EnumEM, AutoClass and GAXEM on five real world data sets. $K$ and $N$ indicate the optimal number of clusters and the number of data items in a data set, respectively. ‘Attributes’ indicates the attributes used in the experiments. ‘BIC’ indicates the average BIC values, ‘Accu’(%) is the average classification accuracy value and ‘Suc’ indicates the successful runs within 10 runs. The unit for the execution time is second. ............................................................... 141

A.1 Experimental results of ESOM, the Budinich’s SOM, SA, and the enhanced CEN when applied 5 benchmark TSPs. An enhanced version is that a network is improved by the $NII$ heuristic. ........... 169
A.2 Experimental results of ESOM, SA, the Budinich’s SOM, the KNIES-local and the KNIES-global on 15 benchmark TSPs. . . . . . . . 172

B.1 13 alleles of an individual and their domains in the neural-evolutionary system, and the parameter setting in the evolved ISOM (eISOM). The learning parameters $\eta_1(t)$, $\eta_2(t)$ and $\sigma(t)$ are decreased linearly after each learning iteration. $\eta_1(t)$ reaches zero at the last loop. . 182

B.2 Experiment results of the SA approach, the Budinich’s SOM, the ESOM, and the evolved ISOM (eISOM), and the enhanced CEN algorithm when applied to 5 TSP benchmarks. . . . . . . . . . . 188

B.3 Experiment results of the SA approach, the FLEXMAP algorithm, the Budinich’s SOM, the ESOM, and the evolved ISOM (eISOM) when applied to the third set of TSPs. . . . . . . . . . . . . . . . 190
List of Figures

1.1 Data mining is an essential step in the process of KDD. .................. 3

2.1 A mixture model for a data set with 6 clusters in a 2-D space. ... 17

2.2 Illustration of model-based clustering of the data set depicted in Figure 2.1. A dot indicates a data item. An ellipse and its associated ‘o’ indicate a contour and the center of a Gaussian distribution component. .................................................. 18

3.1 A scalable model-based clustering framework. ......................... 28

3.2 A mixture model generated from the 72 summary statistics the data set depicted in Figure 2.1. A ‘*’ indicates a summary statistics. The mixture model generated is indicated by the six solid ellipses and their associated ‘o’. The original mixture model is indicated by the six dotted ellipses and their associated ‘+’. .... 30

4.1 Illustration of the first data set. ................................. 37

4.2 Data summarization based on two different grid structures. A ‘*’ indicates a non-empty subcluster. ................................. 38

4.3 The hash function in the adaptive grid-based data summarization procedure. ................................. 39

4.4 Search an appropriate entry in the DS-array. ....................... 42

4.5 An architecture of a self-organizing map. ........................... 44
4.6 A schematic view of two different learning rules (a) in the SOM; (b) in the ESOM. A black disc indicates an input data item and a grey disc indicates a neuron. The dashed circle indicates the interim neuron while a circle indicates a neuron after learning. The dot indicates the origin. .................................................. 45

4.7 Two SOMs from a 2-dimensional space onto 1-dimension one. . . 47

4.8 The quantization error during the learning of the SOM and the ESOM algorithms. .................................................. 54

4.9 Illustration of three data sets. .................................................. 56

4.10 The topological error during the learning of the SOM and the ESOM algorithms. .................................................. 57

4.11 A schematic view of two different learning rules (a) in the elastic net; (b) in the ISOM. A black disc indicates an input data item and a grey disc indicates a neuron. A dashed circle indicates an interim neuron while a circle indicates a neuron after learning. The dot indicates the origin. .................................................. 60

4.12 Illustration of the ISOM for data summarization. A ring of neurons are indicated by black discs and the data items are indicated by the dots. .................................................. 61

5.1 Illustration of the fourth synthetic data set. Only 5% data items are plotted. .................................................. 76

5.2 The clustering features and cluster centers generated by BIRCH for the fourth synthetic data set. A dot indicates a clustering feature, while an ‘x’ denotes a cluster center. .................................................. 77

5.3 The clusters generated by the gEMACF algorithm for the fourth synthetic data set. A ‘*’ indicates a clustering feature. An ‘o’ and an ellipse indicate a generated Gaussian component, while a ‘+’ and a dotted ellipse indicate an original Gaussian component. . . 78
5.4 The execution times on eight 4-dimensional data sets. 81
5.5 Sensitivity of clustering algorithms to the data summarization or sampling procedures. 83
5.6 A typical mixture model generated by the gEMACF algorithm from 8*8 grid structure. The 64 `*' indicate the 64 clustering features. A ellipse and its associated ‘o’ indicate a Gaussian component generated, and a dotted ellipse and its associated ’+’ indicate an original Gaussian component. 84
5.7 A typical mixture model generated by the gEMACF algorithm from 8*128 grid structure. The 748 `*' indicate the 748 clustering features. A ellipse and its associated ‘o’ indicate a Gaussian component generated, and a dotted ellipse and its associated ’+’ indicate an original Gaussian component. 85
5.8 A typical mixture model generated by the bEMACF algorithm for the fourth data set. A dot indicates a clustering feature. An ‘o’ and an ellipse indicate a Gaussian component generated, while a ‘+’ and a dotted ellipse indicate an original Gaussian component. 88
5.9 Execution times of five clustering algorithms for eight 4-dimensional data sets. 89
6.1 The California housing data in a 2-D space. A dot indicates a data item. 98
6.2 A typical mixture model generated by EMACF from 783 clustering features for the 2-D California housing data. A Gaussian distribution component is indicated by an ‘o’ and its associated ellipse. Each `*' represents a clustering feature. 99
6.3 A typical mixture model generated by EMADS for the 2-D California housing data. A Gaussian distribution is indicated by an ‘o’ and its associated ellipse. Each `*' represents a data summary. 100
6.4 Data samples of the first four data sets. (a) A 10% sample of the first data set, (b) A 5% sample of the second data set, (c) A 5% sample of the third data set, and (d) A 10% sample of the fourth data set. ............................................................... 113

6.5 The sensitivity of clustering algorithms to the data summarization or sampling procedure. ............................................................... 114

6.6 Two typical mixture models generated by the gEMADS algorithm from different data summarization procedures. An ‘o’ and its associated ellipse represent a generated Gaussian distribution component. A ‘+’ and its associated dotted ellipse indicates an original Gaussian distribution component. A ‘*’ represents a data summary. ............................................................... 115

6.7 Execution times of the four clustering algorithms for eight 4-dimensional data sets ............................................................... 119

7.1 The flow chart of the genetic-guided model-based clustering algorithms. ............................................................... 130

7.2 Illustration of the HAC crossover operator. A Gaussian distribution component’s parameters consist of $p_k$, $\mu_k$ and $\Sigma_k$. ................. 132

7.3 A typical synthetic data set with its their original mixture model. A solid ellipse and its associated ‘+’ indicate the contour and the mean of a Gaussian component distribution. ............................................................... 135

7.4 Typical mixture models for dataSetI generated by (a). EnumEM, (b). AutoClass, (c). GAEM and (e). GAXEM. The minimal and the average $BIC$ values during a typical run of (d). GAEM and (f). GAXEM are depicted respectively. ............................................................... 137

8.1 The cluster structure in the 2-D California housing data. Seven clusters are indicated by seven ellipses and their associated ‘o’. A dot indicates a data item and an ‘x’ indicates an outlier. ............ 144
8.2 The six clusters and outlier in a data set. An outlier is indicated by
an ‘x’. Six clusters are indicated by six ellipses and their associated
‘o’. ................................................................. 146

A.1 A Schematic view of a SOM for the TSP. In this appendix, \( p = 2 \) . 160
A.2 The convex hull property of a simple TSP: (a) The convex hull
and an optimal tour; (b) A tour without the convex hull property.
The dotted lines in (a) indicate the convex hull. ................. 162
A.3 Experimental results on a set of 18 synthetic TSPs: (a) Compari-
son of the solution qualities yielded by the ESOM, the Budinich’s
SOM and the SA approach; (b) The average execution time com-
parison between the Budinich’s SOM and the ESOM. ............ 167
A.4 Typical solutions obtained for the TSP Att532 with 532 cities: (a)
by the SA approach; (b) by the Budinich’s SOM; (c) by the ESOM.170

B.1 A neural-evolutionary system that evolves learning schemes. ... 179
B.2 (a). Comparison of the solution quality of the evolved ISOM, the
ESOM, the Budinich’s SOM and the SA approach on 18 random
TSPs; (b). The average execution time comparison among the
three SOMs. ...................................................... 186
B.3 4 typical tours of the random TSP with 2400 cities obtained by
(a). the SA approach, (b). the Budinich’s SOM, (c) the ESOM,
(d) the evolved ISOM. ..................................... 192
Chapter 1

Introduction

In this chapter, we first introduce the background of data mining in Section 1.1. Then we focus on cluster analysis in Section 1.2. We give the definition of scalable clustering algorithms. We list the main challenges of cluster analysis in data mining and give a comprehensive review of clustering techniques in data mining. In Section 1.3, the motivation and the contributions of the thesis are presented. We give the organization of the thesis in last section.

1.1 Data mining

Data mining refers to extracting or mining knowledge from large amounts of data. It is becoming one of the most active and exciting research areas. Data mining is a natural result of the evolution of information technology. Our capabilities of both generating and collecting data have been increasing rapidly in the last several decades. Contributing factors include the widespread use of bar codes for most commercial products, the computerization of many businesses, scientific, and government transactions, and the advances in data collection tools ranging from scanned text and image platforms to satellite remote sensing systems. In addition, the popular use of World Wide Web as a global information system has flooded us with tremendous amount of data and information. It is impractical for human to look through all the data and discover some untapped valuable
patterns. We are drowning in data, but starving for knowledge. This explosive growth in stored data has generated an urgent need for new techniques and tools that can intelligently and automatically assist us in transforming the data into useful knowledge.

Data mining can discover valuable information and patterns. The information and patterns can be applied to business management, business decision making, production control, marketing analysis, engineering design, and science exploration. Its wide applications bring data mining to the forefront of business. According to Dan Vesset, research director for market research firm IDC, the worldwide data mining product market grew from US$455 million to US$539 million last year, and it is expected to increase continuously to US$1.85 billion in 2006 [58]. All these trends have been attracting researchers from areas such as database, visualization, artificial intelligence, and statistics [29, 41]. Thus, data mining is considered as one of the most promising interdisciplinary developments in information technology.

Data mining is also viewed as an essential step in the process of Knowledge Discovery in Databases (KDD). It is defined as a non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from large amount of data. Since data mining is an essential and crucial step of KDD, it is also treated as a synonym for KDD to some people. KDD as a process is depicted in Figure 1.1 and consists of an iterative sequence of the following steps [29]:

1. Data selection where data relevant to the analysis task are retrieved from the database,
2. Preprocessing where data are cleaned and/or integrated,
3. Data transformation where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations,
4. Data mining which is an essential process where intelligent methods are
Figure 1.1: Data mining is an essential step in the process of KDD.

applied in order to extract patterns and knowledge, and

5. Interpretation/evaluation which identifies truly interesting patterns representing knowledge based on some interestingness measures.

Data mining technologies are characterized by intensive computations on large amounts of data. The two most significant challenges in data mining are scalability and performance. For an algorithm to be scalable, its running time should grow linearly in proportion to the size of the database, given the available system resources such as main memory and disk space. Data mining functionalities include the discovery of concept/class descriptions, association, classification, prediction, clustering, trend analysis, deviation analysis, and similarity analysis. However, this thesis only concentrates on scalable cluster analysis.

1.2 Cluster Analysis

Clustering is the process of grouping a set of data items (observations, feature vectors, or objects) into classes or clusters so that data items have high similarity when compared with one another within a cluster, but are very dissimilar to
items in other clusters. Unlike in classification, the class label of each data item is previously unknown.

Clustering has wide applications. It is often used as a stand-alone data mining tool to observe the characteristics of each cluster and to focus on a particular set of clusters for further analysis. In business, clustering helps marketers discover distinct groups in their customer bases and characterize customer groups based on purchasing patterns [29]. In biology, it is used to derive plant and animal taxonomies, categorize genes with similar functionality, and gain insight into structures inherent in populations [7, 63]. It can also help to classify documents on the Web for information discovery [16]. Clustering not only can act as a stand-alone tool, but also can serve as a preprocessing step for other algorithms which would then operate on the detected clusters. Those algorithms such as characterization and classification have wide applications in practice. All of them have attracted researchers from many disciplines, including statistics, machine learning, pattern recognition, databases, and data mining.

1.2.1 Challenges of Cluster Analysis in Data Mining

Clustering is a challenging research field where its potential applications pose their own special requirements. In data mining, research efforts have been focused on exploring methods for efficient and effective cluster analysis in large databases. The followings are the typical challenges of clustering in data mining [41].

1. **Scalability**: It concerns how to efficiently handle large databases which may contain millions of data items. For an algorithm to be scalable, its running time should grow linearly in proportion to the size of the database, given the available system resources such as main memory and disk space [10, 33]. Sometimes re-scan the data on servers may be an expensive operation since data are generated by an expensive join query over potentially distributed data warehouse. Thus, only one data scan is usually required.
2. **Discovery of clusters with arbitrary shape:** Many clustering algorithms determine clusters based on Euclidean or Manhattan distance measures. Algorithms based on such distance measures tend to find spherical clusters with similar size and density. It is important to develop algorithms that can detect clusters of arbitrary shape.

3. **Sound theoretic support:** Real world applications require clustering algorithms with statistically sound algorithm, rather than pure heuristics.

4. **Insensitivity to the order of input data:** Some clustering algorithms generate different clusters with different ordering. It is important to develop algorithms insensitive to the order of input.

5. **Minimal requirements for domain knowledge to determine input parameters:** Many clustering algorithms require users to input certain parameters in cluster analysis. These parameters are often hard to determine. This not only burdens users, but also makes the quality of clustering difficult to control.

6. **Ability to deal with noisy data:** Many real world databases contain outliers, missing, unknown or erroneous data.

7. **Ability to deal with different types of attributes:** The types of data may be numerical, binary, categorical (nominal) and ordinal, or mixtures of these data types.

8. **High dimensionality:** Many clustering algorithms are good at handling low-dimensional data, involving only two or three dimensions. However, a data set can contain many dimensions or attributes. It is challenging to cluster data sets in a high-dimensional space because such data can be very sparse and highly skew.
9. **Interpretability and usability**: Users expect clustering results to be interpretable, comprehensible, and usable. That is, clustering may need to be tied up with specific semantic interpretations and applications. It is important to study how an application goal may influence the selection of clustering methods.

Our proposed clustering algorithms in the thesis concentrate on the first six requirements, especially the first four. We are planning to enhance our algorithms to satisfy all requirements in the near future.

### 1.2.2 A Survey of Cluster Analysis in Data Mining

There exist many clustering algorithms in the data mining literature. In general, major clustering methods can be classified into the following categories.

**Partitioning methods**: Given a data set with $N$ data items and $K$, the number of clusters to form, a partitioning algorithm organizes the data into $K$ partitions, where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as distance. Partitioning methods usually use iterative relocation techniques that attempt to improve the partitioning by moving objects from one group to another. The most well-known and commonly used partitioning methods are K-means, K-medoids and their variations [46, 93]. Recently, there are several scalable partitioning methods, which tend to find spherical clusters with similar size and density.

**The scalable K-means algorithm** is based on the idea of dynamically identifying, via K-means, three kinds of regions in data: regions that are compressible, regions that are discardable, and regions that must be maintained in main memory. To clear out main memory for coming data items and the next K-means to be invoked, the data in the first two kinds of
regions are summarized and only their sufficient statistics are maintained in the main memory for the coming K-means [12].

The density-biased sampling K-means algorithm is another scalable K-means variation [72], which assigned high weights for samples from dense regions and vice versa.

CLARANS (Clustering Large Applications based upon RANdomized Search) [69], a scalable clustering algorithm based on K-medoids, searches locally for new medoids around the old ones rather than globally. To reduce the random sampling biases, CLARANS needs to sample the whole data set in batches, and thus its computational complexity is about $O(N^2)$.

Hierarchical methods: A hierarchical method creates a hierarchical decomposition of the given data [50]. They can be further classified into agglomerative and divisive hierarchical clustering, depending on whether the hierarchical decomposition is formed in a bottom-up or top-down fashion. The quality of a pure hierarchical method suffers from its inability to perform adjustment once a merge or a split decision has been executed. One promising direction for improving the clustering quality of hierarchical methods is to integrate hierarchical clustering with other clustering techniques for multiple phase clustering.

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) introduces two concepts, Clustering Feature (CF) for description of a sub-cluster of data items, and CF-tree for incremental and dynamic clustering of the coming data. It begins by partitioning data hierarchically using the CF-tree structure, and finally applied other clustering algorithms to refine the clusters. The computational complexity of the algorithm is $O(N)$ [97]. The underlying idea of CF-tree has been extended to complicated data sets [21, 34]. However, since each node in a CF-tree does not always correspond to what a user may consider a natural cluster. Moreover, if the clusters are
not spherical in shape, BIRCH does not perform well since it uses notion of radius or diameter to control the boundary of a cluster [72, 41].

CURE (Clustering Using REpresentatives) represents each cluster by a fixed number of representative data items and then shrinks them towards the center of the cluster by a specified fraction [38]. With linear complexity, CURE produces high-quality clusters in the existence of outliers, allowing clusters of complex shapes and different sizes. But CURE does not handle categorical attributes.

ROCK [39] merges two subclusters based on their interconnectivity. It is suitable for clustering categorical attributes. ROCK ignores information about the closeness of two subclusters while emphasizes their interconnectivity.

Chameleon takes into account of both the interconnectivity as well as the closeness of the clusters, especially the internal characteristics of the subclusters themselves [49]. However, the processing cost for high-dimensional data may require $O(N^2)$ for $N$ data items in the worst case.

**Density-based methods:** Density-based methods regard clusters as dense regions of data that are separated by regions of low density of data (representing noise). They can discover clusters with arbitrary shape.

**DBSCAN** (Density-based Spatial Clustering of Applications with Noise), the density-based method developed by Ester et al, grows regions with sufficiently high density into clusters and can discover clusters of arbitrary shape in spatial databases with noise [26]. If a spatial index is used, the computational complexity of DBSCAN is $O(N \log N)$. To enhance DBSCAN, Sander et al have developed **GDBSCAN** [76]. GDBSCAN generalizes the notion of point density and therefore it can be applied to objects of arbitrary data type, e.g. 2-dimensional polygons. **BRIDGE** speeds up DBSCAN by working on the partitions generated by K-means [22]. These
algorithms are sensitive to two user-defined density parameters.

**OPTICS** (Ordering Points To Identify the Clustering Structure) is proposed to reduce the sensitivity of density-based clustering methods [4]. It computes an augmented clustering ordering for automatic and interactive cluster analysis. This ordering represents the density-based clustering structure of the data. OPTICS has the same run-time complexity as that of DBSCAN.

**Data Bubbles** speeds up OPTICS dramatically by working on specific compact data objects: data bubbles, which contain summary statistics and distance distribution of subclusters [13].

**DENCLUE** (DENsity-based CLUstErIng) models the overall density of the data space as the sum of influence functions of all data items. Clusters can then be determined mathematically by identifying density attractors, where density attractors are local maxima of the density function. DENCLUE has a solid mathematical foundation and has good clustering properties for data sets with large amounts of noise [43]. However, the selection of two density parameters may significantly influence the quality of the clustering results.

**Grid-based methods:** This kind of methods quantize the object space into a finite number of cells that form a grid structure on which all of the operations for clustering are performed. The main advantage of this kinds of methods is their fast processing time, which is typically independent of the number of data items, yet dependent on only the number of cells in each dimension in the quantized space. But the performance usually degenerates as the dimensionality increases. Some typical examples of the grid-based approach include **STING**, which explores statistics information stored in the grid cells [88]; **WaveCluster**, which clusters objects using a wavelet transform method [78]; and **CLIQUE**, which represents a grid-
and density-based approach for clustering in high-dimensional data space [2].

**Neural networks:** For clustering, neurons in a neural network act as prototypes of clusters. A data item is assigned to its most similar neuron based on some distance measure [42, 48]. The neural network approach to clustering has strong theoretical links with actual brain processing. The **scalable parallel Self-Organizing Map** (SOM) is an attempt to scale-up neural networks [57]. Further research is required in making it readily applicable to larger databases due to long processing time and the intricacies of complex data.

**Model-based methods:** Model-based methods hypothesize a model for clusters and find the best fit of the given model to data. A model-based algorithm may locate clusters by constructing a density function that reflects the spatial distribution of the data points. It also leads to a way of automatically determining the number of clusters based on standard statistics, taking ‘noise’ or outliers into account and thus yielding robust clustering methods. **CLASSIT** [35], a typical example of conceptual clustering, creates a hierarchical clustering in the form of a classification tree. The classification tree is not height-balanced for skew input data, which may cause the time and space complexity to degrade dramatically. The **Expectation-Maximization** (EM) algorithm [23, 64, 91], and its variations like the CEM algorithm [17, 18] and the lazy EM algorithm [83], iteratively fit a mixture model to a data set. **AutoClass** [20], a popular clustering method, uses Bayesian statistical analysis to estimate the number of clusters. These algorithms usually assume the data set maintained in main memory and need to scan through the whole data set multiple times, thus they are not suitable for large data sets. There is some pioneering work to scale-up model-based methods.

**The incremental EM algorithm** updates its mixture model for each
single data item instead of maintaining the whole data set in main memory [68]. The algorithm is sensitive to data input ordering.

**The mrkD-tree based EM algorithm** uses a KD-tree to cache sufficient statistics of interesting regions of data first. Then it applies the EM algorithm on the appropriate KD-tree nodes. However, the speedup factor decreases significantly as dimensionality increases [67].

**The SEM algorithm** (the Scalable EM algorithm) uses a similar idea as the scalable K-means algorithm to dynamically identify three kinds of regions of data. It empirically modifies the classical EM algorithm to accommodate the sufficient statistics of the compressible regions [10]. Compared with the EM algorithm, the SEM algorithm runs 3 times faster and can generate clustering results with little loss of accuracy. However, the SEM algorithm is sensitive to the data input ordering and users have to determine several problem specific parameters.

### 1.3 The Thesis

#### 1.3.1 Motivation

Model-based clustering methods can describe clusters with a variety of shapes. The model-based methods provide us a framework to handle the complicated data set with different kinds of attributes. Furthermore, model-based clustering methods have solid mathematical foundations from the statistics community [64]. They can take ‘noise’ or outliers into account and thus are very robust. Some standard statistics are provided for automatically determining the number of clusters. Therefore, they have been successfully applied to many real world applications. For example, they have been used to analyze astronomic data [20] and basket data [71], to cluster gene expression microarray data [63], and to discover Web navigation patterns [15]. They also have been used in database query
optimization [77].

However, the expensive model construction prohibits them from handling large real world data sets, especially when they are too large to fit into main memory. The speedup factor of recent research efforts is not significant [10, 83]. Till now, there lack good scalable methods for large data sets. The research progress on model-based clustering methods lags behind the other kinds of clustering methods. So, the variety of cluster description power, sound foundations, wide applications, and ubiquitous large real world data sets urge us to develop some scalable model-based clustering algorithms. They can employ limited computation resources to handle large data sets.

1.3.2 Thesis Statement

Working on summary statistics of subclusters, rather than individual data items directly, is a sound and effective strategy to scale-up model-based clustering techniques. It is mathematically sound because our two proposed algorithms guarantee to converge. It is effective because the two algorithms embody the sufficient statistics of each subcluster and then make cluster analysis on data sets with no or little loss of clustering accuracy.

1.3.3 Contributions of the Thesis

The main contributions of the thesis are listed as follows:

1. It gives a multi-phase framework for scalable model-based cluster analysis.

2. It proposes a model-based clustering algorithm for clustering features - EMACF (the EM Algorithm for Clustering Features).
   - It guarantees to converge to a local maximum.
• It is used to establish two scalable clustering systems, the gEMACF and the bEMACF algorithms, by combining with the adaptive grid-based data summarization and the BIRCH’s data summarization procedures respectively.

• Both the gEMACF and the bEMACF algorithms can generate clustering results with no or little loss of accuracy and run one or two orders of magnitude faster than the classical clustering algorithms on both synthetic and real world data sets.

3. It proposes a model-based clustering algorithm for data summaries - EMADS (the EM Algorithm for Data Summaries).

• It uses a new data summarization representation scheme.

• It guarantees to converge.

• It is used to establish two scalable clustering systems, the gEMADS and the bEMADS algorithms by combining with the adaptive grid-based data summarization and the BIRCH’s data summarization procedures respectively.

• The bEMADS algorithm runs one or two orders of magnitude faster than the classical algorithm with no or little loss of clustering accuracy on both synthetic and real world data sets.

4. It proposes an Expanding Self-Organizing Map (ESOM) and Integrated Self-Organizing Map (ISOM) for data summarization and projection, which can also generate the most accurate tours for the travelling salesman problem in the neural network literature so far.
1.4 Thesis Roadmap

In this chapter, we have given the research background of the thesis. We have listed the main challenges and the major research progress of cluster analysis in data mining. The remaining of the thesis concentrated on scalable model-based clustering algorithms.

In Chapter 2, we give the background of the model-based clustering techniques which paves the way for us to develop scalable model-based clustering algorithms.

In Chapter 3, based on some up-to-date scalable clustering techniques, a scalable model-based cluster analysis framework is presented. This framework consists of three modules: the data summarization module, the in-memory cluster analysis module and the optional controller. The framework guides us to develop several scalable model-based clustering systems.

In Chapter 4, we describe three different data summarization procedures for the data summarization module. We develop an adaptive grid-based data summarization procedure. Then we discuss how to use the Self-Organization Map (SOM) to sum up a data set better. We propose two new learning rules and establish two new SOMs: the Expanding SOM (ESOM) and the Integrated SOM (ISOM). The two SOMs are not only applicable to data summarization but also to data projection. One of their typical applications is discussed in Appendices A and B. Besides these two data summarization procedures, we also review the BIRCH’s data summarization procedure in this chapter.

Chapter 5 concentrates on a new in-memory clustering technique: EMACF. It can generate mixture models from the clustering features directly and effectively. After that, some theoretical analyses of EMACF are presented. Combining with the grid-based and the BIRCH’s data summarization procedures, EMACF may be used to establish two scalable systems: the gEMACF and the bEMACF algorithms. Their performance is examined on both synthetic and real world data sets comprehensively.
Chapter 6 focuses on a new in-memory clustering techniques: EMADS. We give a simple data summary representation scheme for a subcluster of data items, and EMADS can take account of the correlations among data attributes and generate mixture models from the simple data summaries directly. After that, some theoretical analyses of EMADS are presented. Combining with the grid-based and the BIRCH’s data summarization procedures, EMADS is used to build two scalable systems: the gEMADS and the bEMADS algorithms. Finally, Their clustering results on both synthetic and real world data sets are presented and compared with three clustering algorithms.

Chapter 7 discusses a genetic-guided technique for determining the number of clusters automatically. We give some preliminary results. Finally, we discuss the method of scaling up the model selection techniques for large data sets.

In Chapter 8, we briefly discuss two other applications of our scalable model-based techniques: outlier detection and database query optimization. After that, we conclude the thesis on its contributions and limitations. Finally, we point out some research directions of the thesis.

In Appendix A, we present the Expanding SOM (ESOM) for the Traveling Salesman Problem (TSP) as an example of ESOM for data projection. The solutions generated by the ESOM are compared with four existing neural networks.

In Appendix B, we first develop an Integrated SOM (ISOM) for the TSP as an example of SOM for data projection. Then we design the ISOM using a genetic algorithm. The evolved ISOM (eISOM) is examined and compared with four neural networks on three sets of TSPs.
Chapter 2

Model-based Cluster Analysis

In this chapter, we give the background of model-based clustering techniques. We describe what a finite mixture model is and how to evaluate it in Section 2.1. In Sections 2.2, 2.3 and 2.4, we review several methods of generating mixture models. We also discuss how to select an appropriate mixture model in Section 2.5.

2.1 Mixture Model and Clustering

2.1.1 Mixture Model

Given a data set $X = (x_1, x_2, \cdots, x_N)$, the model-based clustering algorithms assume that each data item $x_i = [x_{i1}, x_{i2}, \cdots, x_{iD}]^T$ is drawn from a finite mixture model $\Phi$ of $K$ distributions:

$$ p(x_i | \Phi) = \sum_{k=1}^{K} p_k \phi(x_i | \theta_k). \tag{2.1} $$

Here $N$ is the total number of the data items; $D$ is the dimensionality of the data set; $K$ is the number of clusters (or component distributions); $p_k$ is the mixing proportion for the $k^{th}$ cluster ($0 < p_k < 1$ for all $k = 1, \cdots, K$ and $\sum_{k=1}^{K} p_k = 1$); $\phi(x_i | \theta_k)$ is a component density function where the parameters are indicated by a vector $\theta_k$.  

This thesis concentrates on the case where $\phi(x_i|\theta_k)$ is the multivariate Gaussian distribution, even though the framework used is able to be applied to mixture models for complicated data sets. This Gaussian mixture model has been used with considerable success [10, 20, 30]. On the other hand, as stated by density estimation theory, any distribution can be effectively approximated by a Gaussian mixture model [11]. So, the research on Gaussian mixture models is quite important. In a Gaussian mixture model, the parameter $\theta_k$ consists of a mean vector $\mu_k$ and a covariance matrix $\Sigma_k$. The density function is of the form

$$
\phi(x_i|\theta_k) = \frac{\exp\left\{-\frac{1}{2}(x_i - \mu_k)^T\Sigma_k^{-1}(x_i - \mu_k)\right\}}{(2\pi)^{D/2}\sqrt{|\Sigma_k|^{1/2}}}
$$

(2.2)

where $D$ is the dimensionality of data items.

Thus, a mixture model $\Phi$ includes the mixing proportion, the component density function $\phi$ and parameters involved. Given $\Phi$, one may get a crisp classification by assigning the data items $x_i$ to cluster $k$ if $k = \arg\max_l \{p_l \phi(x_i|\theta_l)\}$. Thus, a model $\Phi$ can be viewed as a solution for the clustering problem. So the
clustering problem is transformed into finding parameters of a mixture model $\Phi$. Figure 2.1 illustrates a Gaussian mixture model for a data set with 6 clusters where six Gaussian distributions are indicated by six ellipses and their associated ‘o’. It is clearly observed that the Gaussian mixture model has high value in the dense data regions. The clustering result based on this Gaussian mixture model is illustrated in Figure 2.2. Now, the problem is how to evaluate a mixture model and how to find a good mixture model for a data set.

### 2.1.2 Likelihood

The description accuracy of a mixture model may be measured by two criteria: likelihood and classification likelihood [64]. The likelihood is the joint probability density for all the data items. It measures to what degree the mixture model
matches the data set. The logarithm of the likelihood takes the form

\[ L(\Phi) = \log \left[ \prod_{i=1}^{N} p(x_i|\Phi) \right] = \sum_{i=1}^{N} [\log p(x_i|\Phi)]. \] (2.3)

In general, it is impossible to solve it explicitly, and iterative schemes, such as the Expectation-Maximization (EM) algorithm, must be employed.

The classification log-likelihood, simplified from the log-likelihood, is calculated according to

\[ L_C(\Phi) = \sum_{k=1}^{K} \sum_{x_i \in C_k} \log \phi(x_i|\theta_k) \] (2.4)

where \( C_k \) indicates the \( k^{th} \) cluster. \( \sum_{x_i \in C_k} \log \phi(x_i|\theta_k) \) denotes the contribution of cluster \( C_k \) to the classification log-likelihood [17]. The log-likelihood is quite close to the classification log-likelihood if clusters are well-separated. The classification log-likelihood is usually easier to calculate.

### 2.2 The General EM Algorithm

To pave the way for deriving our two EM Algorithms: EMACF (the EM Algorithm for Clustering Features) and EMADS (the EM Algorithm for Data Summaries) below, we outline the general procedure of the EM algorithm applied to finite mixture models here.

The general Expectation-Maximization (EM) algorithm, a standard tool in the statistical repertoire, is widely applied to iteratively compute a maximum likelihood estimation. It is profitably applied to incomplete-data problems. One typical example is clustering when cluster labels are viewed as ‘missing’ values. The basic idea of the general EM algorithm is to associate the given incomplete-data problem with a complete-data problem for which the maximum likelihood estimate is computationally tractable.

Suppose that we have a set of ‘incomplete’ data vectors \( \{x\} \) and wish to maximize the likelihood \( p(\{x\} | \Phi) \). Let \( \{y\} \) denote a typical ‘complete’ version of \( \{x\} \),
that is, each vector \( \mathbf{x}_i \) is augmented by the ‘missing’ values so that \( \mathbf{y}_i = (\mathbf{x}_i^T, \mathbf{z}_i^T)^T \). There may be many possible vectors \( \mathbf{y}_i \) in which \( \mathbf{x}_i \) is embedded. For the finite mixture case, \( \mathbf{z}_i \) is naturally a cluster indicator vector \( \mathbf{z}_i = (z_{1i}, z_{2i}, \cdots, z_{Ki})^T \), where \( z_{ki} \) is one if \( \mathbf{x}_i \) belongs to the \( k \)th component and zero otherwise. Let the likelihood of \( \{\mathbf{y}\} \) be \( g(\{\mathbf{y}\} | \Phi) \) whose form is known explicitly so that the likelihood \( p(\{\mathbf{x}\} | \Phi) \) is obtained from \( g(\{\mathbf{y}\} | \Phi) \) by integrating over all possible \( \{\mathbf{y}\} \) in which the set \( \{\mathbf{x}\} \) is embedded. The log-likelihood may be computed as

\[
L(\Phi) = \log p(\{\mathbf{x}\} | \Phi) = \int \log \prod_{i=1}^{N} (\mathbf{x}_i, \mathbf{z}_i | \Phi) d\mathbf{z}.
\] (2.5)

The general EM procedure generates a sequence of estimation of \( \Phi \), \( \{\Phi^{(j)}\} \), from an initial estimate \( \Phi^{(0)} \) and consists of two steps:

1. **E-step**: Evaluate \( Q(\Phi; \Phi^{(j)}) \triangleq E [\log(g(\{\mathbf{y}\} | \Phi))|\{\mathbf{x}\}, \Phi^{(j)}] \), that is,

\[
Q(\Phi; \Phi^{(j)}) = \int \sum_{i=1}^{N} \log(g(\mathbf{x}_i, \mathbf{z}_i | \Phi)) p(\{\mathbf{z}\} | \{\mathbf{x}\}, \Phi^{(j)}) d\mathbf{z}_1 d\mathbf{z}_2 \cdots d\mathbf{z}_n,
\]

the expectation of the complete data log-likelihood, conditional on the observed data, \( \{\mathbf{x}\} \), and the current value of the parameters, \( \Phi^{(j)} \).

2. **M-step**: Find \( \Phi = \Phi^{(j+1)} \) that maximizes \( Q(\Phi; \Phi^{(j)}) \), or at least, \( Q(\Phi^{(j+1)}; \Phi^{(j)}) \geq Q(\Phi^{(j)}; \Phi^{(j)}) \).

The log-likelihood of interest satisfies \( L(\Phi^{(j+1)}) \geq L(\Phi^{(j)}) \). Thus for a bounded sequence of log-likelihood values, \( \{L(\Phi^{(j)})\} \) converges monotonically to some log-likelihood value \( L^* \).

### 2.3 The EM Algorithm

It is worth pointing out that, in the statistics community, the EM algorithm is referred to the general EM algorithm above [64]. The classical EM algorithm in other research fields (such as machine learning, data mining) is actually the EM
algorithm for Gaussian mixture models. We follow the terminology in the data mining community.

Given the number of clusters, the EM algorithm estimates the parameters in the Gaussian mixture model via increasing the likelihood iteratively. The EM algorithm is given as follows:

1. Fix the number of clusters $K$, initialize the parameters in the mixture model: $p_k^{(j)}, \mu_k^{(j)}$ and $\Sigma_k^{(j)}$ ($> 0$) ($k = 1, \cdots, K$), and set the current iteration $j$ to 0.

2. **E-Step**: Given the mixture model parameters, compute $t_{ik}^{(j)}$:

$$t_{ik}^{(j)} = \frac{p_k^{(j)} \phi(x_i | u_k^{(j)}, \Sigma_k^{(j)})}{\sum_{l=1}^{K} p_l^{(j)} \phi(x_i | u_l^{(j)}, \Sigma_l^{(j)})} \quad (2.6)$$

3. **M-Step**: Given $t_{ik}^{(j)}$, update the mixture model parameters from the total $M$ data items for $k = 1, \cdots, K$:

$$p_k^{(j+1)} = \frac{1}{N} \sum_{i=1}^{N} t_{ik}^{(j)} \quad (2.7)$$

$$\mu_k^{(j+1)} = \frac{\sum_{i=1}^{N} t_{ik}^{(j)} x_i}{N \cdot p_k^{(j+1)}} \quad (2.8)$$

$$\Sigma_k^{(j+1)} = \frac{\sum_{i=1}^{N} t_{ik}^{(j)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^T}{N \cdot p_k^{(j+1)}} \quad (2.9)$$

4. If $|L(\Phi^{(j+1)}) - L(\Phi^{(j)})| \leq \varepsilon |L(\Phi^{(j)})|$, set $j$ to $j + 1$ and go to step 2. Here $\varepsilon$ is a small positive number.

As a greedy algorithm, EM never decreases the log-likelihood $L(\Phi)$. Thus it converges to a near optimal solution with high log-likelihood value, and then we can get a maximum likelihood estimation of $\Phi$. The EM algorithm does not guarantee to converge to the global maximum. Furthermore, the convergence
rate may be very slow if the clusters are not well separated or the number of clusters is not properly predefined [73]. Researchers have proposed several improved versions, like the CEM algorithm to maximize the classification log-likelihood iteratively [17] and the re-weighted EM algorithm to speedup the convergence rate [90]. However, they still converge to suboptimal solutions and the performance is very sensitive to the initialization. It can be seen that EM has to scan through each data item in its E-step and M-step. This prohibits EM from handling large data sets, especially when the data sets are too large to be loaded into main memory. The main theme of the thesis is to scale-up EM for large data sets with little loss of clustering accuracy.

2.4 Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) is a stage-wise procedure in which ‘optimal’ pairs of clusters are successively merged. In the model-based HAC algorithm, a pair of clusters with the least loss of the classification log-likelihood is chosen to agglomerate at each stage [21, 30].

Although the resulting classifications are suboptimal, the HAC algorithms are commonly used because they often yield reasonable results and are relatively easy to implement. A major drawback of the HAC algorithms is their computation requirement. The computation and the memory complexities depend quadratically on the number of components in the initial partition, which is usually a set of singleton clusters. Thus, it is impractical to process large data sets directly [73].

2.5 Model Selection

Model selection concerns how many clusters in mixture models. The log-likelihood and the classification log-likelihood only measure the description accuracy of the
Chapter 2 Model-based Cluster Analysis

mixture for the data set. Normally, they increase with the number of clusters, thus they cannot act as the criteria to choose an appropriate model directly.

There are many heuristics to choose an appropriate number of clusters. Some of them are based on some heuristic measures or statistical indices for cluster validity [60, 81, 94]. More recently, information theoretic model selection criteria have gained increasing popularity. Information theoretic criteria are mainly based on minimum description lengths (MDL), Bayes rule and Kullback-Leibler (K-L) distance. MDL-based criteria minimize the number of bits required to represent the data. Criteria based on Bayes rule choose the model that maximizes the probability of the data given a model and prior information. The Bayesian solution, with a sound statistical foundation, calculates the posterior probability of the possible number of clusters for the given data set, prior to the mixture model parameters. A potential difficulty with this approach is the computational complexity of integrating over the parameter space to get the posterior probability. The AutoClass uses various approximations to get around the computational issues [20]. Another set of model selection criteria minimize the K-L distance between the candidate model’s fitness and the generating model’s fitness. A number of model selection criteria have been derived based on approximations to this distance. A classic criterion is the *Bayesian Information Criterion (BIC)*, defined by:

\[
BIC(\Phi) = -2 \cdot L(\Phi) + v(\Phi) \log N
\]

where \(N\) is the total number of data items and \(v(\Phi)\) is the number of free parameters in the mixture model \(\Phi\). It prefers a simpler mixture model to better describe a data set. Experimental results have shown its good performance in practice [8, 30].

The common methodology on model selection is based on the enumeration strategy. Given the number of clusters, the best log-likelihood is estimated by invoking the EM algorithm several times with random initialization. Then, the
$BIC$ values for all possible $K$ compete with one another. The model with the minimal $BIC$ value is chosen to determine the number of clusters in the data set $[8, 30, 90]$. The EM algorithm usually finds suboptimal log-likelihood value. Furthermore, the clusterings in different runs have no communication. Thus, the enumeration model-based clustering algorithm does not work well, especially on complicated large data sets.

### 2.6 Chapter Summary

In this chapter, we have outlined the model-based clustering techniques from the mixture model definition, model search to model selection. This review paves the way for our scalable model-based clustering framework in the next chapter. We have described the principle of the general EM algorithm which helps us to formalize two new EM algorithms in Chapters 5 and 6.
A scalable clustering method in the thesis refers to a method whose running time grows linearly in proportion to the total number of data items for the given main memory. Consider sometimes a data set is located in a distributed database or a data warehouse, a data set scan is also an expensive operation [10, 97]. Thus, a scalable clustering algorithm is only allowed to read the data set from disk once. That is to minimize the execution time required for I/O. Besides the execution time, the clustering method also aims to generate as accurate results as possible.

In this chapter, we first review the scalable clustering techniques in Section 3.1. After that, we give and discuss our scalable model-based clustering framework.

### 3.1 Scalable Clustering Techniques

As shown in the survey on the cluster analysis in data mining in Section 1.2.2, scalable cluster analysis has been developed very quickly during the past few years. We now categorize these methods again based on the scaling-up techniques involved.

**Sampling:** Random sampling is one of the most intuitive methods to scale up an algorithm as done in CURE [38] and Chameleon [49]. To reduce the biases
introduced by random sampling, the density biased sampling technique has been proposed by assigning high weights to samples from dense regions, and vice versa [72]. CLARANS [69] and the incremental EM algorithm [83] sample data items in a batch mode. They take a subset of data items and learn from them, and then take a new subset. The procedure continues until reading through the whole data set. The sampling techniques can be integrated with any clustering algorithms. However, they sometimes generate biased clustering results.

Indexing Techniques: Some indexing techniques are used to speed up cluster analysis. They normally vary from one clustering algorithm to another. In DBSCAN [76], a spatial index, say, R*-tree, is used to search for the density-reachable data items. The mrkD-tree based EM algorithm [67] establishes a hierarchical KD-tree to help the EM algorithm to avoid calculation on nonsignificant regions and then speed up the EM algorithm. The density-biased sampling K-means algorithm [72] uses a hash table to index the dense regions.

Restricted Search: Substituting global search by some local search may scale-up a clustering algorithm but sacrifice the accuracy. CLARANS [69] changes only one medoid rather than K medoids in the K-medoids algorithm for each iteration. BRIDGE [21] restricts DBSCAN to run within a subset partitioned by the K-means algorithm.

Data Condensation: Usually, this is a kind of multiple phase clustering techniques. They normally have some specific data structures to sum up or to describe a subcluster of data items.

BIRCH [97] represents a subcluster of data items with a Clustering Feature (CF). Data items are incrementally stored into a hierarchical CF-tree. When the given main memory is used up, the CF-tree can be rebuilt by
increasing the granularity of CF-nodes and then free some main memory. Finally, it uses HAC to generate clusters from the CF-nodes. A similar idea has been used in BUBBLE [34] for data sets in arbitrary metric spaces and ECF (Extended CF) [21] for data sets with mixed attributes.

Data Bubbles [13] stores the statistics as well as the distribution of some nearest neighbor distances of a subcluster of data items in its compact data object: data bubble, which enables OPTICS to process large data sets quickly.

STING [88] establishes a hierarchical grid tree to store the sufficient statistics, the maximum, the minimum, and the type of distribution of data items in a cell and then cluster the data set based on stored information.

Instead of these hierarchical data summarizations, the scalable K-means algorithm [12, 27] and the Scalable EM (SEM) algorithm [10] identify the compressible regions using the K-means and the EM algorithms dynamically. They invokes the K-means and the EM algorithms multiple times and compressible regions are identified and condensed into compact data objects. The scalable K-means algorithm represents a region using its weight center for the subsequent runs of the K-means algorithm. The SEM algorithm condenses a compressible region into the sufficient statistics of data items in this region. In order to accommodate the sufficient statistics, the EM algorithm in the SEM algorithm is empirically modified into the ExEM (Extended EM) algorithm.

The research efforts on scalable model-based clustering algorithms, especially on the EM algorithm, lag behind those of other clustering directions. The mrkD-tree-based EM algorithm can handle low-dimensional data sets efficiently, but its speedup decreases dramatically as the dimensionality increases [67]. The lazy EM algorithm speeds up the EM algorithm two or three times [83], but it has to scan the whole data set within each iteration. The SEM algorithm can generate quite
accurate results even for the given main memory, but it needs to invoke the EM or the ExEM algorithms multiple times [11, 10]. Compared with the EM algorithm, its speedup factor is up to seven. In addition, its core, the ExEM algorithm, lacks sound mathematics support [11]. The incremental EM algorithm can speed up the EM algorithm two to four times [83], but its clustering accuracy is not as good as the SEM algorithm [10]. The underdevelopment situation and the wide applications motivate us to establish several scalable model-based clustering algorithms.

3.2 Multi-phase Model-based Clustering Framework

Our scalable model-based clustering framework consists of three modules. Its basic idea is given as follows. It first partitions a data set into mutually exclusively subclusters. Each subcluster only contains similar data items. Some summary statistics of subclusters are retained in main memory. This actually is a data condensation or data summarization module. After that, a mixture model is generated from the summary statistics directly using some in-memory model-based clustering techniques. The in-memory clustering techniques take into account of the data distribution within subclusters in their iterative procedure and can generate mixture models with no or little loss of the description accuracy. We
term it the in-memory clustering module. Furthermore, to avoid user parameter input, the in-memory clustering algorithms may be invoked a number of times to determine the number of clusters automatically. This is optional and is referred to as the optional controller module in the framework. The interaction of the three modules is illustrated in Figure 3.1.

Now, let us give an illustration on how the framework works. For the data set shown in Figure 2.1, we may simply partition the data space into 9*8 cells, and the data items within the same cell comprise a subcluster and are condensed into a summary statistics. These 72 subclusters are indicated by ‘*’ in Figure 3.2. Based on the 72 summary statistics, our in-memory clustering techniques generate a mixture model. In Figure 3.2, an ‘o’ and its corresponding solid ellipse indicate the mean and the contour of a Gaussian distribution generated. And a ‘+’ and its corresponding dotted ellipse indicate the mean and the contour of an original Gaussian distribution. It is obvious that the generated mixture model is very close to the original one.

3.2.1 Data Summarization

This module is to partition a data set roughly into subclusters and sum up each subcluster into a compact data object: a summary statistics. Thus, the module mainly concerns the best summary statistics for each subcluster and how to efficiently partition the data set into subclusters by scanning the whole data set once. Some scalable clustering algorithms may be used to partition a data set efficiently and we will discuss them in detail in Chapter 4. We discuss two kinds of appropriate summary statistics in this section.

The summary statistics of a subcluster has two basic requirements. The first one is that it should reflect the data distribution within the subcluster. Since a Gaussian distribution takes account of the covariance matrix, this kind of information had better be embedded in the summary statistics. The second requirement
Figure 3.2: A mixture model generated from the 72 summary statistics the data set depicted in Figure 2.1. A ‘*’ indicates a summary statistics. The mixture model generated is indicated by the six solid ellipses and their associated ‘o’. The original mixture model is indicated by the six dotted ellipses and their associated ‘+’.

is that the summary statistics should be generated and operated conveniently. In order to satisfy the requirement that the whole data set is scanned only once, the summary statistics of a subcluster can be updated incrementally when a new data item comes. Furthermore, during the data summarization procedure, we adjust the granularity of subclusters adaptively in order to keep as more summary statistics as possible in the given main memory. At the beginning, the granularity is small, and the data set is partitioned into a lot of subclusters. With the input of data items, the number of summary statistics stored in the main memory become larger and larger. If the limited amount of the main memory is used up, we need to increase the granularity of the subclusters and then reduce the number of summary statistics. In order to avoid re-reading the data items, we may sum up a couple of summary statistics into a new summary statistics. In
other words, a couple of subclusters are merged into a new subcluster. Therefore, several summary statistics should be merged into a new summary statistics.

We discuss two kinds of summary statistics to summarize a subcluster of data items. When data attributes are correlated, we keep the covariance information in our summary statistics.

**Definition 3.1** Given a subcluster, the data summary is defined as a triple: 
\[ DS_m = \{n_m, \nu_m, \Gamma_m\} \] 
where 
\[ M \] 
\[ n_m \] is the number of data items in the \( m^{th} \) subcluster;
\[ \nu_m = \frac{1}{n_m} \sum_{\text{subcluster}} x_i \] is the mean of the data items in the \( m^{th} \) subcluster;
\[ \Gamma_m = \frac{1}{n_m} \sum_{\text{subcluster}} x_i^T x_i \] is the average of the outer products of the \( n_m \) data items.

Basically, the data summary \( DS_m \) comprises the zeroth, the first, and the second moments of the \( m^{th} \) subcluster. It contains sufficient statistics if the data items within the subcluster follow a Gaussian distribution, since the covariance matrix can be calculated by \( n_m \Gamma_m - n_m \nu_m \nu_m^T \). For the second requirement, we have the following additivity theorem.

**Theorem 3.2** Assume that \( DS_p = \{n_p, \nu_p, \Gamma_p\} \) and \( DS_q = \{n_q, \nu_q, \Gamma_q\} \) are two data summaries of two disjoint subclusters, then the data summary \( DS_m = \{n_m, \nu_m, \Gamma_m\} \) of the subcluster that is formed by merging the two disjoint subclusters is:

\[ n_m = n_p + n_q, \quad (3.1) \]
\[ \nu_m = \frac{n_p \nu_p + n_q \nu_q}{n_m}, \quad (3.2) \]
\[ \Gamma_m = \frac{n_p \Gamma_p + n_q \Gamma_q}{n_m}. \quad (3.3) \]
We can prove the theorem true using the definition of the data summary. The theorem is also applicable when a data item is merged with a subcluster. So, we can generate data summaries incrementally.

The second kind of summary statistics is useful when data attributes are statistically independent. The independence assumption is often made in the model-based cluster analysis.

**Definition 3.3** For the $m^{th}$ subcluster, the **clustering feature** is defined as a triple: $s_m = \{n_m, \nu_m, \gamma_m\} \ (m = 1, \cdots, M)$, where

- $n_m$ is the number of data items in the $m^{th}$ subcluster;
- $\nu_m = \frac{1}{n_m} \sum_{\text{the } m^{th} \text{ subcluster}} x_i$ is the mean of the data items in the $m^{th}$ subcluster;
- $\gamma_m = (\gamma_{1m}, \gamma_{2m}, \cdots, \gamma_{Dm})^T = \frac{1}{n_m} \sum_{\text{the } m^{th} \text{ subcluster}} x_i \otimes x_i$ where $\otimes$ is the array multiplication operation. It is the square sum of the $n_m$ data items divided by $n_m$.

In fact, $\gamma_m$ is the diagonal vector of the matrix $\Gamma_m = \frac{1}{n_m} \sum_{\text{the } m^{th} \text{ subcluster}} x_i^T x_i$. So, the clustering feature $s_m$ is a simplification of the data summary $DS_m$. The clustering feature contains the zeroth, the first, and the second moments of the subcluster when data attributes are statistically independent. It is sufficient when the subcluster of data items follow a Gaussian distribution. Similarly, we have the following additivity theorem for the second requirement.

**Theorem 3.4** Assume that $s_p = \{n_p, \nu_p, \gamma_p\}$ and $s_q = \{n_q, \nu_q, \gamma_q\}$ are two clustering features of two disjoint subclusters, then the clustering feature $s_m =$
\( \{n_m, \nu_m, \gamma_m\} \) of the subcluster that is formed by merging the two disjoint subclusters is:

\[
\begin{align*}
    n_m &= n_p + n_q, \\
    \nu_m &= \frac{n_p \nu_p + n_q \nu_q}{n_m}, \\
    \gamma_m &= \frac{n_p \gamma_p + n_q \gamma_q}{n_m}.
\end{align*}
\]

The theorem is a natural result from the definition of the clustering feature. It is worth noting that the clustering feature definition is similar with the one first appears in BIRCH [97]. In fact, the second and the third items in the latter are not divided by the cardinality of the subcluster.

These two kinds of summary statistics may sum up a data set sufficiently for model-based cluster analysis. They can reduce the scale of the original clustering problem. We will discuss how to generate summary statistics in Chapter 4.

### 3.2.2 In-memory Model-based Clustering

This module is used to generate mixture models from the summary statistics directly. The main concern here is how to make better use of the summary statistics to generate accurate clustering results. For two kinds of summary statistics: clustering features and data summaries, we establish two special EM algorithms based on the principle of the general EM algorithm. We will introduce EMACF (the EM Algorithm for Clustering Features) in Chapter 5, and EMADS (the EM Algorithm for Data Summaries) in Chapter 6, respectively.

### 3.2.3 Optional Controller

The first two modules can construct the scalable model-based clustering algorithms if the number of clusters is pre-defined. This module is optional and guides the first two modules to work more intelligently. The module may benefit
the communication between the data summarization and the in-memory clustering modules. For example, it can carry the output of the in-memory clustering module back to the data summarization module as feedback. According to the feedback, the data summarization module may provide better summary statistics to the in-memory clustering module. The controller may facilitate the interaction between the end users and our clustering systems. This module may help to detect the more sophisticated structures hidden in data sets, for example, to determine the appropriate number of clusters in the data set automatically. In Chapter 7, we will introduce some methods to determine the number of clusters automatically.

3.3 Features

According to the above scalable model-based clustering framework, we may establish some scalable model-based clustering systems which have some interesting features.

- They can generate summary statistics by scanning the whole data set only once. The computation complexity of the data summarization module can be as low as linear. The number of summary statistics is much less than the number of data items for large data set. In addition, the number of subclusters is not strictly dependent on the size of the data set, we may adjust the granularity of subclusters and enable the summary statistics to fit into the given main memory.

- The iterative in-memory clustering procedure only works on the summary statistics, and its computational complexity may be as low as $O(M)$, where $M$ is the number of summary statistics. Thus, the computational complexity of the whole model-based algorithm is linear with respect to the number of data items.
• The iterative in-memory clustering procedure considers the data distribution within subclusters explicitly. It enables our clustering algorithms to generate clustering results with no or little loss of accuracy. It also makes our in-memory clustering algorithms less sensitive to the data summarization procedures.

• These in-memory clustering algorithms are derived from the general EM algorithm such that we may establish the sound statistical foundations readily.

• The optional controller module can help to determine the number of clusters for the end users.

• The techniques for handling noisy data in other clustering algorithms may be simply integrated into the data summarization procedures to make our model-based clustering systems more robust.

3.4 Chapter Summary

We have reviewed the scalable clustering techniques in the data mining literature. We have also presented a scalable model-based framework, which guides us to develop some scalable model-based clustering systems. In the following chapters, we expand the framework in detail. We discuss three data summarization procedures in the next chapter. We develop two new in-memory clustering algorithms in Chapters 5 and 6, respectively. The optional controller module is discussed in Chapter 7.
Chapter 4

Data Summarization Procedures

In this chapter, we propose two data summarization procedures with linear computational complexity for the given main memory and review one classical data summarization procedure. We give an adaptive grid-based data summarization procedure in Section 4.1 and a self-organization map data summarization procedure in Section 4.2. The classical data summarization procedure in BIRCH is reviewed in Section 4.3. We mainly discuss how to partition a data set and sum up subclusters into data summaries. All the procedures are also applicable to generate clustering features since a clustering feature is a simplification of a data summary.

4.1 Adaptive Grid-based Data Summarization

The grid-based data summarization procedure partitions a data set by imposing a multidimensional grid structure on the data space and then sum up the data items within a cell into its summary statistics. The similar data items within a cell compose a subcluster. For simplicity, each attribute is partitioned into several equal-width segments by grids. Thus, each cell has the same width for each attribute and has the same volume. For the data set depicted in Figure 4.1, we partition each attribute into 20 segments as shown in Figure 4.2(a). So, the cell widths specify the grid structure and the total number of cells.
To operate within the given main memory, we only store the data summaries for the non-empty cells into a data summary array: $DS$-array. The array has a fixed number of entries, $M$, according to the given amount of main memory. Each entry corresponds an element of the $DS$-array. Besides the data summary, $\{n_m, \nu_m, \Gamma_m\}$ of a non-empty subcluster as defined in Section 3.2.1, each entry also contains a cell ID number, $c_m$, which indicates the associated cell. So, the $m^{th}$ entry is defined as $\{c_m, n_m, \nu_m, \Gamma_m\}$. All data items within a cell can decide the same cell ID according to their attribute value and current grid structure. When a new item is read in, we get which cell it should be in. Then we search for its associated entry in the $DS$-array efficiently by using a hash function which specifies which entries we should search. Once an appropriate entry is found, its data summary is updated to absorb the new data item. The entry found for the data item should either be empty or has been occupied by the corresponding cell.

Figure 4.3 gives a procedure to calculate the cell ID number $c_0$ for a data item $x_i$ according to the current grid width $\mathbf{w}$. This procedure also generate $l_1$ hash indices which can map cells uniformly onto the $DS$-array.
Figure 4.2: Data summarization based on two different grid structures. A '*' indicates a non-empty subcluster.

Figure 4.4 describes how to search for the appropriate entry. Basically, the procedure checks the entries pointed by the hash indices one by one. If all these entries have been occupied by other cells, it checks the subsequent $l_2$ entries after the last hash index one by one.

To make better use of the given main memory, our grid-based data summarization procedure determines the cell width adaptively. At the beginning, the cell widths are initialized to be very small. If the cell widths are too small, the number of non-empty cells may be larger than the number of entries in the $DS$-array. Then the cell widths increase, and the number non-empty cells decreases. The number of non-empty cells gradually fits into the given amount of the main memory. However, during the procedure, the $DS$-array has to be rebuilt.

In order to avoid reading through the data set again for rebuilding the data summary array $DS$-array, we merge pairs of cells into bigger ones such that the new data summaries can be calculated from the old data summaries based on the $DS$ additivity theorem. Our strategy is to double the smallest cell width each
PROCEDURE hashFun(w, x_min, x_i, l_1, l_2, c_0, h);
       // w is the cell-width-vector;
       // x_min contains the minimal values of attributes;
       // x_i is the current input data item;
       // l_1 and l_2 are two constants;
       // c_0 indicates the ID cell number corresponding to x_i;
       // Hash index vector h contains l_1 elements;
       c_0=0;
       FOR d = 1 TO D DO
           c_0 = c_0 * 65,599 + CEIL((x_{di} - x_{d,min})/w_d);
       ENDFOR
       FOR i = 1 TO l_1 - 1 DO
           h_i = MOD(c_0 - 1, M - i) + 1;
       ENDFOR
       h_{l_1} = MOD(c_0 - 1, M - l_2 - 1) + 1;

Figure 4.3: The hash function in the adaptive grid-based data summarization procedure.

time. The granularity of the cell side on this attribute becomes larger. A new cell exactly comprises two old cells, and the number of the non-empty cells becomes smaller. Thus, the DS-array may be rebuilt to make more entries available. For the grid structure shown in Figure 4.2(a), we double the cell width for the second attribute, and get the grid structure shown in Figure 4.2(b). Then the 334 non-empty cells are merged into 185 cells, and a data summary of a new cell is calculated from a pair of old data summaries.

During the rebuilding, each entry in the DS-array is treated in the same way as a data item, where the cell ID number is calculated according to the mean vector. The function in Figure 4.3 assures that a proper pair of cells in the old grid structure have the same cell ID number in the new grid structure and then be merged effectively.

We outline the adaptive grid-based data summarization procedure as follows:

Algorithm 4.1 (Adaptive Grid-based Data Summarization Procedure)
1. Initialize a data summary array $DS$-array according to the given main memory. The cell ID number of each entry, $c_m$, is initialized to -1 to indicate that the entry is not occupied by any cell;

2. Initialize the cell-width-vector $w$ according to

$$w_d = \frac{x_{d,\text{max}} - x_{d,\text{min}}}{\text{CEIL}(\frac{x_{d,\text{max}} - x_{d,\text{min}}}{w_0})}$$

where $w_0 = \min_d x_{d,\text{max}} - x_{d,\text{min}}$, $x_{\text{max}}$ and $x_{\text{min}}$ contains the maximal the minimal values of each attribute respectively. So, the total number of cells is around $2^D M$ and the cell widths are close to one another;

3. Read in a data item $x_i$; calculate its cell ID number $c_0$ and hash-index-vector $h$ by invoking $\text{hashFun}(w, x_{\text{min}}, x_i, l_1, l_2, c_0, h)$;

4. Invoke $\text{searchEntry}(DS\text{-array}, h, c_0, l_1, l_2, m, \text{success})$; if find an appropriate entry ($\text{success}==1$), go to Step 6;

5. Adjust the grid structure and rebuild the $DS$-array as follows:

   (a) Allocate and initialize another data summary array $DS$-array2;

   (b) Double the minimal element of the cell-width-vector $w$;

   (c) For each entry $\{c_m, n_m, \nu_m, \Gamma_m\}$ in the $DS$-array,

      i. Invoke $\text{hashFun}(w, x_{\text{min}}, \nu_m, l_1, l_2, c_m, h)$;

      ii. Search an appropriate entry in the $DS$-array2 by invoking $\text{searchEntry}(DS\text{-array2}, h, c_m, l_1, l_2, m_2, \text{success})$;

      iii. Merge $\{c_m, n_m, \nu_m, \Gamma_m\}$ into $\{c_{m_2}, n_{m_2}, \nu_{m_2}, \Gamma_{m_2}\}$ in $DS$-array2;

   (d) Release $DS$-array; $DS$-array2 $\Rightarrow$ $DS$-array;

   (e) Calculate a new cell ID number $c_0$ and a hash-index-vector $h$ for $x_i$ by invoking $\text{hashFun}(w, x_{\text{min}}, x_i, l_1, l_2, c_0, h)$, go back to Step 4.

6. Assimilate $x_i$ by merging $\{c_0, 1, x_i, x_i^T x_i\}$ into $\{c_m, n_m, \nu_m, \Gamma_m\}$. 
7. If there are still some data items left, go to Step 3.

The mergence procedure maintains the data summaries using the DS additivity theorem. To merge an entry \( \{c_k, n_k, \nu_k, \Gamma_k\} \) into another entry \( \{c_m, n_{m_2}, \nu_{m_2}, \Gamma_{m_2}\} \), we use the following formulae:

\[
\begin{align*}
  c_m &= c_k \\
  \nu_m &= \frac{n_{m_1} \nu_{m_1} + n_k \nu_k}{n_m + n_k} \\
  \Gamma_m &= \frac{n_{m_1} \Gamma_{m_1} + n_k \Gamma_k}{n_m + n_k} \\
  n_m &= n_m + n_k.
\end{align*}
\]

(4.1)

In our implementation, the number of entries of DS-array \( M \) is set to \( \min\{30^D, 4000\} \) based on some preliminary experiment results. So, the required main memory may be as small as to store \( M \) data summaries. The parameter \( l_1 \) is set to 3, i.e., we use three different hash indices. The parameter \( l_2 \), the maximal subsequent entries for linear search, is set to 37.

For each data item, we only need to calculate the \( l_1 \) hash indices and search for at most \( l_1 + l_2 \) entries if an appropriate entry is found. So, the computation complexity is \( O(D) \). At the beginning, we have at most \( 2^D M \) cells, and each adjustment of the grid structure decreases the number of cells by a factor of 2. There are up to \( D \) times of adjustment. Within each adjustment, there are at most \( M \) entries to calculate. In summary, the computation complexity of the grid-based data summarization procedure is \( O(ND) \). On the other hand, the memory requirement is to store the two DS-arrays with \( M \) entries each. So, the space complexity is \( O(MD) \) for clustering features and \( O(MD^2) \) for data summaries, respectively.

There are two reasons for using a grid-based data summarization procedure. The first one is that its implementation is easy, especially for low-dimensional data sets. The second one is that the grid structures can be manipulated easily to generate some extremely skew summary statistics for examining the sensitivity of our new model-based clustering algorithms.
4.2 SOM for Data Summarization

When using a Self-Organization Map (SOM) for data summarization, each neuron is regarded as a subcluster center. In this section, we propose two new learning rules for better data summarization.

4.2.1 Self-Organization Map

The Kohonen’s Self-Organizing Map (SOM) has been proved useful as data clustering and projection tools [54]. It uses a small set of well-organized neurons
to represent a data set and creates a neighborhood preserving mapping from a high-dimensional data space onto a low-dimensional grid of neurons. These neurons are located on a low-dimensional, usually 2-dimensional, grid which provides us a convenient surface to project and summarize data. Interestingly, the mapping roughly preserves the most important topological and metric relationships of data, and thus, inherent clusters in data [54, 66, 84].

Figure 4.5 illustrates the basic architecture of the Self-Organizing Map (SOM). The SOM consists of two layers of neurons, indicated by discs. The input neurons receive data \( x_k(t) = [x_{1k}(t), x_{2k}(t), \cdots, x_{Dk}(t)]^T \in \mathbb{R}^D \) \((1 \leq k \leq N)\) at time \( t \) where \( D \) is the dimensionality of the data space and \( N \) is the number of data items in the data set. The output neurons are located on the grid with certain neighborhood relationship. Normally, a rectangular or hexagonal neighborhood is used [84]. In this section, we use the rectangular neighborhood as illustrated in Figure 4.5. The synaptic weights between the \( j^{th} \) output neuron and all input neurons form a vector \( w_j(t) = [w_{1j}(t), w_{2j}(t), \cdots, w_{Dj}(t)]^T \in \mathbb{R}^D \) \((1 \leq j \leq M)\), where \( M \) is the number of output neurons. Therefore, these output neurons have two topologies. One is in the data space, indicated by the weight vector \( w_j(t) \). The other lies on the grid. The underlying idea of the SOM is to construct a topology preserving map by mapping from the \( D \)-dimensional data space onto the 2-dimensional grid.

The SOM establishes a projection through an iterative learning procedure. It behaves like a flexible net that folds onto the ‘cloud’ formed by the input data during the learning. In each learning step, one data item \( x_k(t) \) is chosen randomly, and the distances between it and all the weight vectors of the output neurons are calculated using a certain distance metric (e.g., the Euclidean distance). A neuron \( m(t) \), whose weight vector is closest to the input data \( x_k(t) \), is chosen as the winning neuron (or best-matching unit):

\[
m(t) = \arg\min_j \|x_k(t) - w_j(t)\|^2
\]
where $\|\cdot\|$ denotes the distance measure. Then, the winning neuron and its adjacent neurons on the grid are trained using the input data item. In the data space, they move closer to the input data as illustrated in Figure 4.6(a), where a black disc indicates a data vector, a grey disc indicates a neuron, a solid line indicates the neighbor relationship on the grid, a circle indicates the new position of a neuron, and a dashed arrow indicates a movement direction. Mathematically, their weight vectors are updated according to

\[
    w_j(t + 1) = w_j(t) + \epsilon(t) h_{j,m(t)} [x_k(t) - w_j(t)].
\]

Here $\epsilon(t)$ is the learning rate and usually shrinks to zero. The neighborhood function $h_{j,m(t)}$ may be specified as

\[
    h_{j,m(t)} = \exp \left[ -\frac{[DIS(j,m(t))]^2}{2\sigma(t)^2} \right]
\]

where $DIS(j,m(t))$ is the distance between neurons $j$ and $m(t)$ on the grid, and $\sigma(t)$ is the neighbor width parameter. The learning rule changes the weight vectors of the output neurons so that they become more representative among the
Figure 4.6: A schematic view of two different learning rules (a) in the SOM; (b) in the ESOM. A black disc indicates an input data item and a grey disc indicates a neuron. The dashed circle indicates the interim neuron while a circle indicates a neuron after learning. The dot indicates the origin.

input data items. On the other hand, the cooperative learning among adjacent neurons allows the SOM to construct a neighborhood preserving map so that the neurons adjacent on the grid have similar weight vectors. That is, these neurons are close to one another in the data space. The cooperative learning is a prominent feature of the SOM and its variants.

The SOM’s learning procedure pursues two goals simultaneously [54]. The first goal is adequate vector quantization. Because the number of neurons usually much less than the number of input data items, we may choose the most representative neurons to describe the data set. From the viewpoint of clustering, it is a vector quantization problem. The quantization error $E_Q$ can be used to evaluate how well the weight vectors (or neurons) represent the data set [51, 85]. It is specified as follows:

$$E_Q = \frac{1}{N} \sum_{k=1}^{N} \| x_k(t) - w_{m_k}(t) \|$$  \hspace{1cm} (4.4)

where $m_k$ is the winner for the data vector $x_k(t)$. In fact, this is the objective of
the K-means algorithm if the neurons are regarded as centers of clusters. Figure 4.7(a) depicts a set of 10 neurons to represent 100 data items, where the connected dots indicate a string of neurons, and other dots indicate data. Thus, the data items nearest to a neuron may compose a subcluster. We can draw summary statistics of the data set from these subclusters. Then the quantization error can measure how good the data summarization is.

The second goal of the SOM’s learning is the satisfactory preservation of topology from the data space onto the grid of neurons. The cooperative learning among adjacent neurons enables the SOM to detect and preserve the neighborhood relationship between the two spaces so that neurons adjacent on the grid have similar weight vectors in the data space. However, because the SOM projects the data from a high-dimensional space onto a low-dimensional space which is usually 2-dimensional, a dimensional conflict may occur and a perfect topology preserving mapping may not be generated [6, 51]. For example, the trained SOM shown in Figure 4.7(b) folds the neuron string onto data irregularly. It gets different topology information from the one shown in Figure 4.7(a). The topological error $E_T$ used in [51, 86] may measure to what degree the topological information is retained in SOMs. The topological error $E_T$ is defined as the proportion of the data items for which the closest and the second-closest neurons are not adjacent on the grid. For the map shown in Figure 4.7(a), the topological error is 0, which indicates that it retains better topology than its counterpart in Figure 4.7(b).

The topological and the quantization errors may conflict in the SOM. We will propose two new SOMs in Subsections 4.2.2 and 4.2.3, respectively. They improve data summarization and projection in terms of the two criteria simultaneously.

### 4.2.2 Expanding SOM

Besides the neighborhood relationship in the conventional SOM, another topology relationship can be detected and preserved during the learning procedure to
achieve a better topology preserving mapping for data projection and visualization. This is a linear ordering relationship based on the distance between data and their center. A SOM algorithm can preserve this ordering relationship if the larger the distance between a data item and the center of all data items is, the farther away the corresponding output neuron from the center of all neurons is.

Our Expanding SOM (ESOM) is proposed to preserve both the neighborhood and the ordering relationships. Since this mapping preserves more topology information of the input data, better performance can be achieved.

We introduce a new learning rule to learn the linear ordering relationship. Different from the SOM’s learning rule given in Eq.(4.2), the ESOM’s learning rule has an additional factor, the expanding coefficient $c_j(t)$, which is used to push neurons away from the center of all data items during the learning procedure. In other words, the neuron net is expanding gradually during training. Moreover, the expanding force is specified according to the ordering of the data items. In general, the larger the distance between the corresponding data item and the center, the larger the expanding coefficient $c_j(t)$ is. Consequently, the associated output neuron is pushed away from the center and the ordering of data items is thus preserved in the output neurons. In the following, the ESOM will be given first and then its theoretical analysis will be discussed. Finally, we will present some experimental results for the ESOM.

Figure 4.7: Two SOMs from a 2-dimensional space onto 1-dimension one.
The ESOM Algorithm

Our ESOM algorithm is given below followed by some explanation.

**Algorithm 4.2 (ESOM)**

1. Linearly transform the coordinates $x'_i = [x'_{1i}, x'_{2i}, \ldots, x'_{Di}]^T$ $(i = 1, \ldots, N)$ of all given data items so that they lie within a sphere $S_R$ centered at the origin with radius $R$ ($< 1$). Here $N$ is the number of data items, $D$ is the dimensionality of the data set. Hereafter, $[x_{1i}, x_{2i}, \ldots, x_{Di}]^T$ denotes the new coordinate of $x_i$. Let the center of all data items be $x'_C = \frac{1}{N} \sum_{i=1}^{N} x'_i$ and the maximum distance of data from the center of the data be $d_{\text{max}}$, then

$$x_i = \frac{R}{d_{\text{max}}} (x'_i - x'_C) \text{ for all } i. \quad (4.5)$$

2. Set $t = 0$, and the initial weight vectors $w_j(0)$ $(j = 1, 2, \ldots, M)$ with random values within the above sphere $S_R$ where $M$ is the number of output neurons.

3. Select a data item at random, say $x_k(t) = [x_{1k}, x_{2k}, \ldots, x_{Dk}]^T$, and feed it to the input neurons.

4. Find the winning output neuron, say $m(t)$, nearest to $x_k(t)$ according to the Euclidean metric:

$$m(t) = \arg \min_j \| x_k(t) - w_j(t) \|^2. \quad (4.6)$$

5. Train neuron $m(t)$ and its neighbors using the following formula:

$$w_j(t + 1) = c_j(t)w_j'(t + 1) \overset{\triangle}{=} c_j(t) \{ w_j(t) + \alpha_j(t) [x_k(t) - w_j(t)] \}. \quad (4.7)$$

The parameters include:

- *the interim neuron* $w_j'(t+1)$, which indicates the position of the excited neuron $w_j(t)$ after moving towards the input data item $x_k(t)$;
• the learning parameter $\alpha_j(t) \in [0, 1]$, which is specified by a learning rate $\epsilon(t)$ and a neighborhood function $h_{j,m(t)}$:

$$\alpha_j(t) = \epsilon(t) \times h_{j,m(t)}; \quad (4.8)$$

• the expanding coefficient $c_j(t)$, which is specified according to

$$c_j(t) = [1 - 2\alpha_j(t)(1 - \alpha_j(t)) \kappa_j(t)]^{-\frac{1}{2}}. \quad (4.9)$$

Here $\kappa_j(t)$ is specified by

$$\kappa_j(t) = 1 - \langle x_k(t), w_j(t) \rangle - \sqrt{(1 - \|x_k(t)\|^2)(1 - \|w_j(t)\|^2)}. \quad (4.10)$$

6. Update the effective neighbor width $\sigma(t)$ and the learning rate $\epsilon(t)$ with predetermined decreasing schemes. If the learning loop does not reach a predetermined number, go to Step 3 with $t := t + 1$.

The first step facilitates the realization of the expanding coefficient $c_j(t)$. After transformation, we can use the norm of a data item $\|x_k(t)\|$ to represent its distance from the center of the transformed data items since the center is the origin. Thus, the norm $\|x_k(t)\|$ can indicate the ordering topology in the data space. This ordering will be detected and preserved in $\|w_j(t)\|$ through the expanding process.

The learning rule defined in Eq.(4.7) is the key point of the proposed ESOM algorithm. Differing from the previously known SOM’s learning rules, it has an additional multiplication factor — the expanding coefficient $c_j(t)$. It is worth pointing out that, although the expanding coefficient $c_j(t)$ is relevant to all data items, $c_j(t)$ only explicitly depends on $\alpha_j(t)$, $x_k(t)$ and $w_j(t)$. If $c_j(t)$ is a constant 1.0, the ESOM algorithm is simplified to a conventional SOM. Since $c_j(t)$ is always greater than or equal to 1.0, the expanding force pushes the excited neuron away from the center. In other words, the inequality $\|w_j(t + 1)\| \geq \|w_j'(t + 1)\|$ is always true. Figure 4.6(b) illustrates the expanding functionality where the
dashed circle indicates the neuron’s interim position and the dot indicates the origin, i.e., the center of the data. After moving the excited neuron \( w_j(t) \) towards the input data item \( x_k(t) \), as indicated by \( w_j'(t+1) \), the neuron is then pushed away from the center. So, during the learning procedure, the flexible neuron net is expanding in the data space. More interestingly, the expanding coefficient \( c_j(t) \) can help us to detect and preserve the ordering relationship.

**Theoretical Analysis**

To show the feasibility of the ESOM algorithm, we should verify that the ESOM algorithm does generate a map that preserve both the good neighborhood and the ordering relationships. In this section, we only give a theorem on a one-step trend to support the feasibility of the ESOM algorithm because it is very difficult to prove the convergence of the SOM-like algorithms in high dimensional cases. In fact, it is still one of the long-standing open research problems in neural networks [62]. We will perform a more rigorous convergence analysis in our future research work. In the following theorem, we assume that all input data items are located within the sphere \( S_R \) and their center coincides with the origin because the preprocessing procedure in Step 1 has been executed.

**Theorem 4.3** Let \( S_R \) be the closed sphere with radius \( R (< 1) \) centered at the origin, \( \{ x_k(t) \in S_R \} \) (for \( k = 1, \cdots, N \)) be the input data and \( \{ w_j(t) \} \) (for \( j = 1, \cdots, M \)) be the weight vectors of the ESOM algorithm at time \( t \). Then, for any \( t \geq 0 \),

(i). for \( j \in \{1, 2, \cdots, M\} \),

\[
1 \leq c_j(t) \leq \frac{1}{\sqrt{1-R^2}}; \quad (4.11)
\]

and \( w_j(t) \in S_R \), that is,

\[
\|w_j(t)\| \leq R. \quad (4.12)
\]
(ii). the expanding coefficient $c_j(t)$ increases with $\|x_k(t)\|$ when $\|x_k(t)\| \geq \|w_j(t)\|$.

**Proof:** (i). We prove Eqs.(4.11) and (4.12) together by induction. This is trivially true for $t = 0$ according to Step 2 of the ESOM algorithm. If we assume that both equations hold for certain $t(\geq 0)$, then we find

$$1 - \kappa_j(t) = \langle x_k(t), w_j(t) \rangle + \sqrt{(1 - \|x_k(t)\|^2)} \sqrt{(1 - \|w_j(t)\|^2)}$$

$$\leq \left( \sum_{d=1}^{D} x_{dk}^2(t) + \left( \sqrt{(1 - \|x_k(t)\|^2)} \right)^2 \right)$$

$$\times \left( \sum_{d=1}^{D} w_{dj}^2(t) + \left( \sqrt{(1 - \|w_j(t)\|^2)} \right)^2 \right)$$

$$= 1.$$

Similarly,

$$1 - \kappa_j(t) = \langle x_k(t), w_j(t) \rangle + \sqrt{(1 - \|x_k(t)\|^2)} \sqrt{(1 - \|w_j(t)\|^2)}$$

$$\geq -\frac{1}{2}(\|x_k(t)\|^2 + \|w_j(t)\|^2) + \sqrt{(1 - R^2)} \sqrt{(1 - R^2)}$$

$$\geq 1 - 2R^2.$$

Thus,

$$0 \leq \kappa_j(t) \leq 2R^2$$

On the other hand, for any learning parameter $\alpha_j(t) \in [0, 1]$, the following inequality is true,

$$0 \leq \alpha_j(t) (1 - \alpha_j(t)) \leq 0.25.$$
According to Eq.(4.9), we get $1 \leq c_j(t) \leq \frac{1}{\sqrt{1-R^2}}$. According to the ESOM’s learning rule, we have

$$1 - \|w_j(t+1)\|^2 = \left[|c_j(t)|^2 - \|w_j(t) + \alpha_j(t)(x_k(t) - w_j(t))\|^2\right] \times (c_j(t))^2$$

$$= \frac{\left(1 - \alpha_j(t)\right)\sqrt{1 - \|w_j(t)\|^2} + \alpha_j(t)\sqrt{1 - \|x_k(t)\|^2}}{(c_j(t))^2}$$

$$\geq \left(1 - \alpha_j(t)\right)\sqrt{1 - R^2} + \alpha_j(t)\sqrt{1 - R^2} \right)^2$$

$$= 1 - R^2. \quad (4.13)$$

This implies that $\|w_j(t+1)\| \leq R$ for any $j = 1, \ldots, M$. Thus, by induction, $w_j(t) \in S_R$ for any $j$ and $t$.

(ii). We rewrite $x_k(t)$ and $w_j(t)$ as follows,

$$x_k(t) = \rho \times e_{x_k}$$
$$w_j(t) = r \times e_{w_j}$$

Here $e_{x_k}$ and $e_{w_j}$ are two unit vectors, and $\rho = \|x_k(t)\|$ and $r = \|w_j(t)\|$. According to the assumption, $\rho \geq r$ holds. Let

$$F(\rho) = \langle w_j(t), x_k(t) \rangle + \sqrt{(1 - \|w_j(t)\|^2)(1 - \|x_k(t)\|^2)} \quad (4.14)$$

$$= \rho \cdot r \cdot \langle e_{w_j}, e_{x_k} \rangle + \sqrt{(1 - \rho^2)(1 - r^2)}.$$  

According to Eq.(4.9), it is obvious that $F(\rho) = 1 - \frac{1 - c_j^{-2}(t)}{2\alpha_j(t)(1-\alpha_j(t))}$. $F(\rho)$ decreases with the expanding coefficient $c_j(t)$. So, to justify the increasing property of $c_j(t)$, it is sufficient to show that $F(\rho)$ decreases with $\rho$ whenever $\rho \geq r$. A direct calculation shows

$$\frac{\partial F(\rho)}{\partial \rho} = r \cdot \langle e_{w_j}, e_{x_k} \rangle - \frac{\rho}{\sqrt{1 - \rho^2}}\sqrt{1 - r^2} \quad (4.15)$$

$$\leq r - \rho \leq 0. \quad (4.16)$$

This implies the decreasing property of $F(\rho)$ on $\rho$ when $\rho \geq r$. \qed
Remarks for the Theoretical Analysis

Theorem 4.3(i) says that the expanding coefficient \( c_j(t) \) is always larger than or equal to 1.0. In other words, it always pushes neurons away from the origin. Thus, during learning, the neuron net is expanding. Furthermore, though the expanding force is always greater than or equal to 1.0, it will never push the output neurons to infinite locations. In fact, it is restricted by sphere \( S_R \) in which the data items are located. This point is substantiated by Eq.(4.12). This supports the feasibility of our ESOM algorithm.

Theorem 4.3(ii) gives a theoretic support that the ESOM algorithm aims to detect and preserve the ordering relationship among the training data items. It points out that the expanding coefficient \( c_j(t) \), or the expanding force, is different for various data items. The larger the distance between a data item and the center of all data items is, the stronger the expanding force will be on the associated output neuron. Consequently, the neuron will be pushed away from the center.

We now briefly discuss another interesting trend based on the proof procedure. If \( w_j(t) \) is far away from \( x_k(t) \), \( \langle e_{w_j}, e_{x_k} \rangle \) will be very small or even less than 0. From Eq.(4.15), \( \frac{\partial F(\rho)}{\partial \rho} \approx -\frac{\rho}{\sqrt{1-\rho^2}} \sqrt{1-r^2} \leq 0 \). In other words, the expanding coefficient \( c_j(t) \) increases with \( \|x_k(t)\| \). So, the ordering of \( \|x_k(t)\| \) is reflected by the expanding coefficient \( c_j(t) \) and then is learned by \( w_j(t) \). This also explains why the topological error of the ESOM algorithm decreases more quickly than that of the SOM algorithm at the beginning of learning. A typical example can be found in Figure 4.8.

Now we derive the computational complexity of the ESOM algorithm. Two differences between the ESOM and the SOM algorithms are Step 1 and the learning rule. The computational complexity of the preprocessing step is \( O(N) \). The learning rule of the ESOM algorithm in Eq.(4.7) needs only a few extra arithmetic operations in comparison with the one of the conventional SOM algorithm.  

\footnote{This is a common case at the beginning of learning since the weight vector \( w_j(t) \) is randomly initialized.}
Figure 4.8: The quantization error during the learning of the SOM and the ESOM algorithms.

Thus, the total computational complexity of the ESOM algorithm is comparable to that of the SOM algorithm which is $O(MN)$ [85].

**Experiments of ESOM for Data Summarization**

We examine the performance of the proposed ESOM algorithm for data summarization and projection which are measured in terms of the quantization and the topological errors respectively. The experiments are conducted on three synthetic data sets as shown in Figure 4.9. All experimental results are compared with those of the SOM algorithm in terms of both the execution time and the mapping quality. All data sets are preprocessed using the same linear transformation as in Eq.(4.5) in order to compare results fairly. All experiments are done using the same set of parameters. The initial values of the learning rate $\epsilon$, the neighbor width $\sigma$, and the radius $R$ are 0.5, 0.9 and 0.999 respectively. Both the learning rate $\alpha$ and the neighbor width parameter $\sigma$ are decreased by a factor of
0.998 per iteration. All experiments are run for 2000 iterations before stopping. We use a rectangular grid with 20*20 neurons.

The three data sets are quite interesting in both their special cluster shapes and locations as illustrated in Figure 4.9. The first data set in Figure 4.9(a) has 3,000 data items in 2-dimensional space with 3 clusters. The inside cluster looks like a triangle which is surrounded by two strip-like clusters. This data set is designed to demonstrate the performance of SOMs, because we can compare the resultant mapping with the original data set directly. The second data set in Figure 4.9(b) contains 2,000 data items in 3-dimensional space. It has two clusters which are on two concentric and spherical layers. They looks like two ‘onionskins’. The third data set in Figure 4.9(c) has 3,000 3-dimensional data items. It contains three clusters on three concentric and spherical layers as illustrated in Figure 4.9(c). These three clusters look like three ‘onionskins’ where the bigger one surrounds the smaller one, and so on.

First, we check the quality of the mappings generated by the SOM and the ESOM algorithms in terms of the two measures. Figures 4.8 and 4.10 illustrate respectively the quantization and the topological errors during a typical run of two algorithms for the first data set. It can be clearly seen from Figure 4.8 that the quantization error decreases gradually as the learning procedure continues. That means that the data summarization becomes more and more accurate. The quantization error of the ESOM algorithm is 0.0175 and the quantization error of the SOM algorithm is 0.0182. During the learning, the topological error curves depicted in Figure 4.10 have three stages: decreasing, increasing and convergence. 

At the very beginning of the learning procedure, the neuron’s weights are fairly dislike, while some of them even contain remnants of random initial values, thus higher topological errors are observed. After several iterations, the topological error decreases dramatically. Because the learning rate $e(t)$ is large and the neighborhood function is also large, the neurons adjacent on the grid may move much closer to the input data item together. At this stage, the ESOM algorithm
Figure 4.9: Illustration of three data sets.
can learn the ordering topology of data items very quickly. As shown in Figure 4.10, the topological error of the ESOM algorithm is much smaller than that of the SOM algorithm. Though the topological errors of both algorithms increase later, the ESOM algorithm keeps the gain and always has smaller topological error than the SOM algorithm. Finally, the topological errors of the ESOM and the SOM algorithms are 0.2381 and 0.3042 respectively. Thus, the ESOM algorithm can generate better mappings than the SOM algorithm for the first data set.

Tables 4.1 and 4.2 list the average quantization and the topological errors of

Table 4.1: Average quantization errors of ESOM and SOM.

<table>
<thead>
<tr>
<th>data set</th>
<th>D</th>
<th>N</th>
<th>ESOM</th>
<th>SOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3000</td>
<td>0.0177</td>
<td>0.0180</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2000</td>
<td>0.0475</td>
<td>0.0483</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3000</td>
<td>0.0599</td>
<td>0.0604</td>
</tr>
</tbody>
</table>
the SOM and the ESOM algorithms for the three data sets. The results are based on 10 independent runs. For the first data set, the average quantization error of the ESOM algorithm is 0.0177, which is 1.7% smaller than the value of 0.0180 for the SOM algorithm. Furthermore, the average topological error of the ESOM algorithm is 0.2409, which is 8.1% smaller than the value of 0.2604 for the SOM algorithm. For the second data set, the quantization errors of the ESOM and the SOM algorithms are 0.0475 and 0.0483, respectively. The average topological errors are 0.3022 and 0.3056, respectively. That is, the ESOM algorithm makes 1.7% improvement on the quantization error and 1.1% improvement on the topological error. For the third data set, the quantization errors of the ESOM and the SOM algorithms are 0.0599 and 0.0604, respectively. The average topological errors are 0.3313 and 0.3476, respectively. That is, the ESOM algorithm makes 0.8% improvement on the quantization error and 4.9% improvement on the topological error. Thus, the ESOM algorithm can generate better mappings than the SOM algorithm for the three data sets. In other words, it can generate better data summarization and projection.

Table 4.3 lists the average execution times in second of the ESOM and the SOM algorithms for the three data sets. On average, the ESOM algorithm takes 3.7%, 0.8%, and 1.1% longer than the SOM algorithm respectively.

In a word, the ESOM algorithm takes slightly longer to generate better mappings than the SOM algorithm. The better topology preserving property of the ESOM algorithm has been successfully applied to data visualization, which may be regarded as a projection from a high-dimensional data space onto a 2 or 3

Table 4.2: Average topological errors of ESOM and SOM.

<table>
<thead>
<tr>
<th>data set</th>
<th>$D$</th>
<th>$N$</th>
<th>ESOM</th>
<th>SOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3000</td>
<td>0.2409</td>
<td>0.2604</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2000</td>
<td>0.3022</td>
<td>0.3056</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3000</td>
<td>0.3313</td>
<td>0.3476</td>
</tr>
</tbody>
</table>
dimensional space. The details of the ESOM algorithm for visualization appear in our recent research work [79].

Another successful application of the ESOM algorithm is to handle the Traveling Salesman Problem (TSP). A solution of the TSP can be viewed as a projection from a 2-D space onto a 1-D space. The linear ordering in the 1-D space can be used to generate a tour of the TSP. The better projection may lead to better tours. Comprehensive experimental results substantiate that the ESOM algorithm can generate more accurate tours than other SOMs. The details of application of the ESOM algorithm to the TSP, and the experiment and comparison results are placed in Appendix A.

4.2.3 Integrated SOM

There are many methods to further enhance the performance of the ESOM algorithm. For example, we can enhance the learning rule and enable a SOM to detect and preserve better topological information. The learning rule in the elastic net, proposed by Durbin and Willshaw [25], is often used to enhance the SOM [3, 45]. This learning rule is illustrated in Figure 4.11(a). It is given as follows:

\[
\begin{align*}
\mathbf{w}_j(t + 1) &= \mathbf{w}_j(t) + \alpha_j(t) [\mathbf{x}_k(t) - \mathbf{w}_j(t)] \\
&\quad + \frac{\beta_j(t)}{2} [\mathbf{w}_{j-1}(t) + \mathbf{w}_{j+1}(t) - 2\mathbf{w}_j(t)] \quad (4.17)
\end{align*}
\]

where \(\beta_j(t)\) is another learning rate parameter. The last expression in the right-hand side of Eq.(4.17) attracts the excited neuron to the middle point of its data set.

<table>
<thead>
<tr>
<th>data set</th>
<th>(D)</th>
<th>(N)</th>
<th>ESOM (Seconds)</th>
<th>SOM (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3000</td>
<td>2371.6</td>
<td>2283.2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2000</td>
<td>2068.9</td>
<td>2051.2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3000</td>
<td>2904.7</td>
<td>2872.1</td>
</tr>
</tbody>
</table>
two neighboring neurons. It reflects the elastic force constraint that reduces the length of the resultant ring of neurons as far as possible [25].

The three learning rules in Eqs. (4.2), (4.7) and (4.17) emphasize different aspects of the SOMs. Thus, we may integrate these ideas together and develop a novel Integrated SOM (ISOM) algorithm to take advantage of these three learning mechanisms. We give a new learning rule as follows:

\[
\mathbf{w}_j(t+1) = c_j(t) \times \left\{ \mathbf{w}_j(t) + \alpha_j(t) [\mathbf{x}_k(t) - \mathbf{w}_j(t)] \right\} \\
+ \frac{\beta_j(t)}{2} [\mathbf{w}_{j-1}(t) + \mathbf{w}_{j+1}(t) - 2\mathbf{w}_j(t)].
\]  

(4.18)

The ISOM algorithm based on this rule is also applicable to data summarization and projection. Figure 4.12 illustrates a set of 3,000 data items and 100 data summaries generated by the ISOM algorithm, where subclusters are indicated by a string of black disks and the data items are indicated by the dots. We has successfully applied the ISOM algorithm to a special case of data projection: the TSP. We enclose the details in Appendix B.
4.3 Data Summarization Procedure of BIRCH

BIRCH (Balanced Iterative Reducing and Clustering Using Hierarchies) is an integrated hierarchical clustering method. It uses the clustering features and Clustering Feature tree (CF-tree) to summarize cluster representations. These structures help the clustering method to achieve good scalability in large databases [97].

A clustering feature is essentially a summary statistics for the given subcluster as introduced in Section 3.2.1. A CF-tree is a height-balanced tree that stores the clustering features for a hierarchical clustering. By definition, a nonleaf node in a tree has children. The nonleaf node store sums of the clustering features of their children, and thus summarize clustering information about their children. A CF-tree has two parameters: branching factor, B, and threshold, T. The branching factor specifies the maximum number of children of a nonleaf node. The threshold
parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters influence the size of the resulting tree.

BIRCH scans the data set to build an initial in-memory CF-tree, which can be viewed as a multilevel compression of the data that tried to preserve the inherent clustering structure of the data. The CF-tree is built dynamically as data items are inserted. Thus, the method is incremental. An data item is inserted to the closest leaf entry. If the diameter of subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split. After insertion of the new data item, information about it is passed toward the root of the tree. The size of the CF-tree can be changed by modifying the threshold. If the size of memory that is needed for storing the CF-tree is larger than the size of the given main memory, then a larger threshold value can be specified and the CF-tree is rebuilt. The rebuilding process is done without rereading all of the data items. This is similar to the insertion and node split in the construction of $B^+$-trees. Therefore, for building the tree, data items have to be read just once. Once all data items are stored in the CF-tree, BIRCH applies a hierarchical agglomerative clustering (HAC) algorithm to cluster the leaf nodes of the CF-tree [97]. BIRCH tries to produce the best clusters with the available resources. The computational complexity of the algorithm is $O(ND)$, where $N$ is the number of data items and $D$ is the data dimensionality. Here we have not included the factors like $B$, $T$, and the memory size.

Experiments have shown the linear scalability of the algorithm with respect to the number of objects, and the good quality of finding clusters. BIRCH usually brought two kinds of critiques. One is that a CF-tree node does not always correspond to what a user may consider a natural cluster. The problem is serious when a partitioning or HAC clustering algorithm is applied on the CF-tree nodes. The second one is that, if the clusters are not spherical in shape, BIRCH does not perform well because it uses the notion of radius or diameter to control the boundary of a cluster. Our in-memory model-based clustering algorithms
discussed in Chapters 5 and 6 can generate some cloud-like clusters from the CF-tree nodes, which may enhance BIRCH to some degrees.
Chapter 5

Scalable Cluster Analysis by Working on Clustering Features

In this chapter, we propose a new algorithm to generate mixture models from clustering features effectively and efficiently in Section 5.1. This new algorithm, termed as EMACF (the EM Algorithm for Clustering Features), is proved to converge to a local maximum. Combining with the grid-based data summarization procedure, we use EMACF to establish a scalable model-based clustering system — the gEMACF algorithm, whose performance is examined in Section 5.2. Combining with the BIRCH’s data summarization procedure, we develop another scalable model-based clustering system — the bEMACF algorithm. We examine its performance comprehensively in Section 5.3.

5.1 EMACF

5.1.1 Motivation

If the attributes are statistically independent, we use clustering features to sum up subclusters. The clustering features $s_m = \{n_m, \nu_m, \gamma_m\}$ $(m = 1, \ldots, M)$ give the zeroth, the first and the second moments of subclusters from a statistical point of view.
For the first attempt, we have tried to generate a mixture model from the clustering features using the classical EM algorithm. Data items within a subcluster are treated in the same way in order to reduce the calculation of both the E-step and the M-step. Basically, each data item is treated in the same way as its associated subcluster mean. It looks like the EM algorithm working on the samples with different weights. The weights are the cardinality of the subclusters. We call it EMAWS (the EM Algorithm for Weighted Samples). We omit its details since it is a simplified version of the algorithm we proposed below.

The clustering accuracy of EMAWS is substantially lower than the sampling EM algorithm, as shown in Section 5.2. So, it is not a good method to treat a data item in the same way as its subcluster mean. We need a better method to process clustering features under Gaussian mixture models.

Our proposed EMACF (the EM Algorithm for Clustering Features) is able to make better use of the clustering features. It is motivated by two interesting observations of the classical EM algorithm for Gaussian mixture models. Let us consider 1-dimensional case first. In EMAWS, we simply use a subcluster mean vector to substitute a data item. For a Gaussian distribution with parameters \( \{\mu_{dk}, \sigma_{dk}\} \), the probability of the data item becomes

\[
\phi(\nu_{dm}|\mu_{dk}, \sigma_{dk}) = \frac{\exp \left[ -\frac{1}{2\sigma_{dk}} (\nu_{dm} - \mu_{dk})^2 \right]}{\sqrt{2\pi\sigma_{dk}}}.
\]

This probability cannot embody the basic characteristic of a Gaussian distribution that a data item in a dense area has a high probability. An intuitive solution is to insert the subcluster variance into the density function. As we know, \( n_m (\gamma_{dm} - \nu_{dm}^2) \) is the variance of the \( m^{th} \) subcluster. We insert the normalized variance into the density function and get:

\[
\phi(\nu_{dm}|\mu_{dk}, \sigma_{dk}) \approx \frac{\exp \left\{ -\frac{1}{2\sigma_{dk}} [(\gamma_{dm} - \nu_{dm}^2) + (\nu_{dm} - \mu_{dk})^2] \right\}}{\sqrt{2\pi\sigma_{dk}}}.
\] (5.1)

Thus the dispersion of the subcluster influences the probability of the associated data items. Smaller \( (\gamma_{dm} - \nu_{dm}^2) \) is, higher the probability is. In other words, the
data within a dense subcluster has a high probability, and vice versa. This point accords with the Gaussian distribution.

Another observation is from the calculation of the M-step of the EM algorithm. If the data items within the $m^{th}$ subcluster have the same probability under each component distribution, so they behave in the same way as in the E-step. In other words, they have the same value of $t_{ik}^{(j)}$ in Eqs.(2.6)-(2.9). It is denoted by $r_{mk}^{(j)}$ below. Let us have a look at the simplification happened in the M-step, especially for the calculation of the cluster variance in Eq.(2.9). We rewrite Eq.(2.9) into

$$Np_k^{(j+1)}\sigma_{dk}^{(j+1)} = \sum_{i=1}^N t_{ik}^{(j)} (x_{di} - \mu_{dk}^{(j+1)})^2 = \sum_{m=1}^M \sum_{x_i \in \text{the } m^{th} \text{ subcluster}} t_{ik}^{(j)} (x_{di} - \mu_{dk}^{(j+1)})^2.$$  

Then $t_{ik}^{(j)}$ is replaced by $r_{mk}^{(j)}$:

$$Np_k^{(j+1)}\sigma_{dk}^{(j+1)} = \sum_{m=1}^M r_{mk}^{(j)} \sum_{\text{the } m^{th} \text{ subcluster}} (x_{di} - \mu_{dk}^{(j+1)})^2 = \sum_{m=1}^M r_{mk}^{(j)} \left[ \sum_{\text{the } m^{th} \text{ subcluster}} \left( x_{di}^2 - 2x_{di}\mu_{dk}^{(j+1)} + n_m (\mu_{dk}^{(j+1)})^2 \right) \right]$$

$$= \sum_{m=1}^M r_{mk}^{(j)} n_m \left[ \gamma_{dm} - 2\nu_{dm} \cdot \mu_{dk}^{(j+1)} + (\mu_{dk}^{(j+1)})^2 \right] \quad (5.2)$$

With the principle of the general EM algorithm given in Section 2.2, we will unify the two observations in our proposed algorithm below.

### 5.1.2 Algorithm Derivation

We aim to generate a Gaussian mixture model $\Phi$ from the clustering features directly. Since we do not know the value of each data item, we cannot use Gaussian density function. We need to define a new density function only based on which subcluster a data item $x_i (\in \mathbb{R}^D)$ belongs to. According to the first observation in Eq.(5.1), our density function is as follows.
**Definition 5.1** For a data item $x_i$ in the $m^{th}$ subcluster, its probability under the probability density function $\psi$ is

\[
\psi(x_i \in \text{the } m^{th} \text{ subcluster}| \theta_k) \triangleq \psi(s_m|\theta_k) \tag{5.3}
\]

\[
= \prod_{d=1}^{D} \exp \left\{ -\frac{1}{2\sigma_{dk}} (\gamma_{dm} - 2\mu_{dk}\nu_{dm} + \mu_{dk}^2) \right\} (2\pi)^{\frac{1}{2}}\sigma_{dk}^{-\frac{1}{2}}.
\]

Here $\theta_k = \{\mu_k, \sigma_k\}$ is the parameters of the $k^{th}$ component density function.

It is worth pointing out that $\theta_k = \{\mu_k, \sigma_k\}$ is also the parameters of the $k^{th}$ component of the Gaussian mixture model $\Phi$. The density function becomes a Gaussian density function when $\gamma_{dm} = \nu_{dm}^2$, especially when there is only one data item in the subcluster. However, this is not a density function in general since its integral is less than 1.0. Under this function, the probability for the data item $x_i$ is implicitly relevant to its values. This point enables us to treat data items within the same subcluster in the same way and we need not store $x_i$. So it helps us to save lots of computation time and main memory. Furthermore, as analyzed above, the dispersion of the subcluster influences the probability of the associated data items. Smaller $(\gamma_{dm} - \nu_{dm}^2)$ is, higher the probability is. In other words, the data within a dense area has a high probability, and vice versa. This point accords with Gaussian mixture models to pay more attention on dense areas.

With this density function, the probability of the data item $x_i$ within the $m^{th}$ subcluster under the mixture model is:

\[
p(x_i \in \text{the } m^{th} \text{ subcluster}|\Psi) \triangleq p(s_m|\Psi) = \sum_{k=1}^{K} p_k \psi(s_m|\mu_k, \sigma_k). \tag{5.4}
\]

Here $\Psi$ indicates a mixture model with component density function $\psi$ in Eq.(5.3). Thus, $\Psi$ contains the same parameters with the Gaussian mixture model $\Phi$, but with a different density function. The log-likelihood of $\Psi$ is

\[
L(\Psi) = \log \left[ \prod_{i=1}^{N} p(x_i|\Psi) \right] = \sum_{m=1}^{M} n_m \log p(s_m|\Psi), \tag{5.5}
\]
where $N$ is the total number of data items.

We now derive an EM algorithm to get the maximum likelihood estimation for the mixture model $\Psi$. The derivation is based on the general EM algorithm by interpreting the cluster labels as missing values. If $x_i$ is in cluster $k$, we denote $z_i$ an indicator vector of length $K$ with 1 in the $k^{th}$ element and zeros elsewhere. Then the complete data vector is $y_i = (x_i^T, z_i^T)^T$. The likelihood of the complete data, $y_i$, is

$$g(y_i|\Psi) = p(x_i|z_i, \Psi)p(z_i|\Psi) = \psi(x_i|\theta_k)p_k = \prod_{k=1}^{K} [\psi(x_i|\theta_k)p_k]^{z_{ki}}.$$ 

The last equation holds since $z_{ki}$ is either zero or one. For the totally $N$ data items, we have

$$g(y_1, \cdots, y_N|\Psi) = \prod_{i=1}^{N} \prod_{k=1}^{K} [\psi(x_i|\theta_k)p_k]^{z_{ki}} = \prod_{m=1}^{M} \prod_{k=1}^{K} [\psi(s_m|\theta_k)p_k]^{z_{kmn_m}}$$

(5.6)

The last equation is valid because that data items within a subcluster have the same probability, and then have same cluster indicator. That is, $\psi(x_i|\theta_k) = \psi(x_j|\theta_k)$ and $z_i = z_j$ if $x_i$ and $x_j$ fall into the same subcluster.

The log-likelihood $L(\Psi)$ is obtained from $g(\{y\}|\Psi)$ by integrating over all possible $\{y\}$ in which the set $\{x\}$ is embedded:

$$L(\Psi) = \log [p(\{x\}|\Psi)] = \int \log [g(\{y\}|\Psi)] dz = \sum_{m=1}^{M} n_m \log p(s_m|\Psi).$$

This agrees with the log-likelihood given in Eq.(5.5).

Now let us calculate the function $Q(\Psi; \Psi^{(j)})$, or the expectation of the complete data log-likelihood, conditional on the observed data $\{x\}$ (which is replaced by $\{s\}$ literally) and the current value of the parameters, $\Psi^{(j)}$.

$$Q(\Psi; \Psi^{(j)}) = E \left[ \log(g(\{y\}|\Psi))|\{x\}, \Psi^{(j)} \right] = E \left[ \log(g(\{y\}|\Psi))|\{s\}, \Psi^{(j)} \right]$$

$$= \sum_{m=1}^{M} n_m \sum_{k=1}^{K} E \left[ z_{km}|\{s\}, \Psi^{(j)} \right] \left[ \log p_k + \log(p(s_m|\mu_k^{(j)}, \sigma_k^{(j)})) \right]$$

$$\triangleq \sum_{i=1}^{M} n_m \sum_{k=1}^{K} \tau_{mk} \left[ \log p_k + \log(p(s_m|\mu_k^{(j)}, \sigma_k^{(j)})) \right]$$

(5.7)
where

\[ r_{mk} = E[z_{km}|\{s\}, \Psi^{(j)}] = \frac{p_k^{(j)} \psi(s_m|\mu_k^{(j)}, \sigma_k^{(j)})}{\sum_{l=1}^{K} p_l^{(j)} \psi(s_m|\mu_l^{(j)}, \sigma_l^{(j)})}. \]  

(5.8)

Here \( r_{mk} \) is the membership probability of a data item in the \( m^{th} \) subcluster belonging to the \( k^{th} \) component.

Now we turn to maximize \( Q(\Psi; \Psi^{(j)}) \) with respect to \( \Psi \). Consider the parameters \( p_k, \mu_k, \) and \( \sigma_k \) in turn. We need to introduce a Lagrange multiplier \( \lambda \) to remove the constraint \( \sum_{k=1}^{K} p_k = 1 \). Differentiating \( Q(\Psi; \Psi^{(j)}) - \lambda \left( \sum_{k=1}^{K} p_k - 1 \right) \) with respect to \( p_k \), we get

\[ \sum_{m=1}^{M} n_m r_{mk} \frac{1}{p_k} - \lambda = 0 \]  

for \( k = 1, \ldots, K \). Summing up the \( K \) equations together, we have

\[ \lambda \sum_{k=1}^{K} p_k = \sum_{k=1}^{K} \sum_{m=1}^{M} n_m r_{mk} = \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} r_{mk} \right] = \sum_{m=1}^{M} n_m = N. \]

This leads to

\[ \lambda = N, \]  

(5.9)

\[ \hat{p}_k = \sum_{m=1}^{M} n_m r_{mk}/N. \]  

(5.10)

For the distribution parameters, we have partial derivative on the density function:

\[ \frac{\partial \log \psi(s_m|\mu_k, \sigma_k)}{\partial \mu_k} = \frac{1}{\sigma_k} (\nu_{dm} - \mu_{dk}), \text{ and} \]

\[ \frac{\partial \log \psi(s_m|\mu_k, \sigma_k)}{\partial \sigma_k} = - \frac{1}{2\sigma_k} + \frac{1}{2\sigma_{dk}^2} (\gamma_{dm} - 2\mu_{dk}\nu_{dm} + \mu_{dk}^2). \]

Differentiating \( Q(\Psi; \Psi^{(j)}) \) with respect to \( \mu_{dk} \) and equating the partial differential to zero gives

\[ \frac{\partial Q(\Psi; \Psi^{(j)})}{\partial \mu_{dk}} = \sum_{m=1}^{M} n_m r_{mk} \frac{1}{\sigma_{dk}} (\nu_{dm} - \mu_{dk}) = 0. \]
This gives the re-estimation for $\mu_{dk}$ as

$$
\hat{\mu}_{dk} = \frac{\sum_{m=1}^{M} n_m r_{mk} \nu_{dm}}{\sum_{m=1}^{M} n_m r_{mk}}.
$$

(5.11)

So, the new cluster center $\hat{\mu}_k$ is a weighted mean of subcluster means. The weight depends on both the cardinality of a subclusters $n_m$ as well as its expected memberships $r_{mk}$. Similarly, differentiating $Q(\Psi; \Psi^{(j)})$ with respect to $\sigma_{dk}$ and equating the result to zero leads to

$$
\hat{\sigma}_{dk} = \frac{\sum_{m=1}^{M} n_m r_{mk} (\gamma_{dm} - 2\mu_{dk} \nu_{dm} + \mu_{dk}^2)}{\sum_{m=1}^{M} n_m r_{mk}}.
$$

(5.12)

It is worth pointing out that the equation accords with the second observation in Eq.(5.2). So, our proposed algorithm unifies the two interesting observations. We rewrite the algorithm in terms of vector as follows.
Algorithm 5.2 (EMACF)

1. **Initialization**: Fixing the number of clusters $K$, initialize the parameters in the mixture model, $p_k^{(j)}(>0), \mu_k^{(j)}$ and $\sigma_k^{(j)}(>0)$ ($k = 1, \ldots, K$), and set the current iteration $j$ to 0.

2. **E-step**: Given the mixture model parameters $\Psi^{(j)}$, compute the membership $r^{(j)}_{mk}$:

$$ r^{(j)}_{mk} = \frac{p_k^{(j)} \psi(s_m | u_k^{(j)}, \sigma_k^{(j)})}{\sum_{i=1}^{K} p_i^{(j)} \psi(s_m | u_i^{(j)}, \sigma_i^{(j)})}. \quad (5.13) $$

3. **M-step**: given $r^{(j)}_{mk}$, update the mixture model parameters for $k = 1, \ldots, K$:

$$ p_k^{(j+1)} = \frac{1}{N} \sum_{m=1}^{M} n_m r^{(j)}_{mk}, \quad (5.14) $$

$$ \mu_k^{(j+1)} = \frac{\sum_{i=1}^{M} n_m r^{(j)}_{mk}\mu_m^{(j)}}{\sum_{m=1}^{M} n_m r^{(j)}_{mk}} = \frac{\sum_{i=1}^{M} n_m r^{(j)}_{mk}\mu_m^{(j)}}{N \cdot p_k^{(j+1)}}, \quad (5.15) $$

$$ \sigma_k^{(j+1)} = \frac{\sum_{m=1}^{M} n_m r^{(j)}_{mk}\left(\gamma_m - 2\mu_k^{(j)} \otimes \nu_m + \mu_k^{(j)} \otimes \mu_k^{(j)}\right)}{N \cdot p_k^{(j+1)}} \quad (5.16) $$

where $\otimes$ indicates the array multiplication. That is, the $(i,j)$ element of $A \otimes B$ is $a_{ij}b_{ij}$.

4. **Termination**: If $|L(\Psi^{(j+1)}) - L(\Psi^{(j)})| \geq \epsilon |L(\Psi^{(j)})|$, set $j$ to $j + 1$ and go to step 2.

With the independence assumption, EMACF is a little different from the classical EM algorithm for Gaussian mixture models. It is also easy to implement.

### 5.1.3 Convergence Guarantee

EMACF is certainly terminated as supported by the following convergence theorem.
Theorem 5.3 $L(\Psi)$ for EMACF converges monotonically to some log-likelihood value $L^* = L(\Psi^*)$ for some stationary mixture model $\Psi^*$.

Proof: We need to prove the following three issues.

1. EMACF is an instantiation of the general EM algorithm, so $L(\Psi)$ for EMACF increases monotonically.

2. $\{L(\Psi)\}$ has an upper bound.

3. The function $Q(\Phi; \Psi)$ is continuous in both $\Phi$ and $\Psi$.

Since $\{p_k\}$ and $\{\sigma_{dk}\}$ are initialized with values larger than zero, they will always be larger than zero according to Eqs.(5.13)-(5.16).

In the derivation procedure above, we follow the general EM algorithm to get EMACF, and thus EMACF is an instance of the EM algorithm. Then we have that $\{L(\Psi^{(j)})\}$ does not decrease, i.e., $L(\Psi^{(j+1)}) \geq L(\Psi^{(j)})$. With Jensen’s inequality,

$$\gamma_{dm} = \frac{1}{n_m} \sum_{x_i \in \text{the } m^{th} \text{ subcluster}} x_{di}^2 \geq \left[ \frac{1}{n_m} \sum_{x_i \in \text{the } m^{th} \text{ subcluster}} x_{di} \right]^2 = \nu_{dm}^2,$$

then

$$\psi(x \in \text{the } m^{th} \text{ subcluster}|\theta_k) = \psi(s_m|\theta_k) \leq \phi(\nu_m|\theta_k).$$

So,

$$L(\Psi) \leq \sum_{m=1}^{M} n_m \log p(s_m|\Psi) \leq \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} p_k \psi(s_m|\theta_k) \right] \leq \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} p_k \phi(\nu_m|\theta_k) \right] < \infty.$$

Thus, we have $L(\Psi^{(j)})$ converges monotonically to some value $L(\Psi^*)$.

The function $Q(\Phi; \Psi)$ in Eq.(5.7) is continuous in both $\Phi$ and $\Psi$, because it consists of only arithmetic and logarithm operations. Recall that the convergence theorem about the EM algorithm, i.e., Theorem 3.2 in [64, p.88], says that all the limit records of any instance $L(\Psi^{(j)})$ of the EM algorithm are stationary records of $L(\Psi)$ provided that the function $Q(\Phi; \Psi)$ is continuous in both $\Phi$ and $\Psi$. The proof of the theorem is completed. \hfill \blacksquare
5.1.4 Complexity

Let us have a brief discussion of the complexity of EMACF. In the E-step, it needs to calculate $M \times K$ membership probabilities $r_{mk}^{(j)}$. Each $r_{mk}^{(j)}$ has to calculate the probability each subcluster mean for each component distribution according to Eq.(5.3). Based on the independent component distribution assumption, it involves $O(D)$ arithmetic operations. Thus the E-step takes $O(MKD)$ operations.

Similarly, the M-step of EMACF takes $O(MKD)$. In a word, the computational complexity of EMACF is $O(MDI)$ where $I$ is the number of iterations. Usually, all EM algorithms are terminated as a constant number of iterations, say, 500 in our implementation, are used up. So, EMACF is linear with the number of subclusters $M$, the number of clusters $K$, and the dimensionality $D$.

EMACF requires $M(2D + 1)$ floating point numbers to store the clustering features, $MK$ floating point numbers for the membership, $K(2D + 1)$ floating point numbers for mixture model. Thus, the total storage requirement of EMACF is $2MD + MK + 2KD + K + M$ floating point numbers. Consequently, the storage requirement of EMACF is independent to the number of data items $N$. In other words, given a data set, we can choose an appropriate $M$ to summarize the data set into the given main memory, and enable the whole clustering system to work within the given main memory.

5.2 Performance of gEMACF

In this section, we illustrate three aspects of EMACF. First, EMACF can be used to construct scalable clustering systems. Secondly, EMACF can generate accurate clustering results in comparison with existing algorithms. Lastly, EMACF is less sensitive to data summarization procedures.
5.2.1 Methodology

Working on the clustering features generated by the adaptive grid-based data summarization procedure discussed in Section 4.1, EMACF can be used to form a scalable clustering system. We term it as the gEMACF algorithm hereafter. There are three reasons for us to choose grid-based data summarization procedure. Firstly, we hope to sum up data set by scanning the whole data set only once and store the clustering features in the given main memory. Secondly, we hope to easily manipulate the shape of subcluster so as to examine the performance of EMACF on some extreme cases. Lastly, it also facilities us to compare EMACF with a density biased sampling clustering algorithm, which is also grid-based [72].

To illustrate the performance of the gEMACF algorithm, we compare it with several sampling or data summarization based clustering algorithms. The algorithms used are listed as follows.

The classical EM algorithm: It is denoted by iEM below. Here we use ‘i’ to denote that attributes are statistically independent.

The sampling EM algorithm: It is iEM working on 5% random samples. It is denoted by sampiEM below.

The Inverse Biased Sampling algorithm: It is called IBS below. It is a typical implementation of density-biased sampling K-means algorithm. It performs better than BIRCH on skew data sets [72]. On average, it draws one weighted data item from a subcluster whose weight is proportional to the cardinality of the subcluster.

the gEMAWS algorithm: EMAWS is our first attempt to generate mixture models from clustering features. Only the first two items in a clustering feature are used. It can be simplified from EMACF by substituting $\gamma_m$ with $\nu_m \otimes \nu_m$ in both the E-step and the M-step. Combining with the adaptive
grid-based data summarization, it is used to establish a scalable clustering algorithm. The algorithm is referred to as the gEMAWS algorithm below.

The gExEM algorithm: The ExEM algorithm has been originally used in [10] to accommodate both data items and sufficient statistics. It explicitly embedded the second moment in its M-step in a heuristic way. Basically, its M-step is similar to the M-step in EMACF while its E-step is identical with iEM [11]. Because the ExEM algorithm is beyond the scope of the general EM algorithm, it is not easy to see whether or not it converges to a local maximum. The gExEM algorithm below is the ExEM algorithm working on the clustering features generated by the adaptive grid-based data summarization procedure. It runs faster than the SEM algorithm in [11] since the latter invokes the ExEM algorithm much more than once.

Our data sets used are quite skew. That is, the cluster sizes vary greatly. In this case, the density-biased sampling algorithm, say, IBS, performs substantially better than BIRCH [72]. Thus, BIRCH is not included in our comparison. All of these algorithms are coded in MATLAB and the experiments are conducted on a Sun Enterprise E4500 server. These model-based clustering algorithms are initialized with the cluster centers generated by the K-means algorithm from the weighted samples. We use two termination criteria for all algorithms. One is the maximal number of iterations, 500. The other is that the successive log-likelihood change is within $10^{-5}$ of current value as used in [65, 83]. In other words, $\epsilon$ is set to $10^{-5}$ in the EM, ExEM and EMACF algorithms. All experimental results reported are averaged on 10 independent runs.

### 5.2.2 Evaluation Metrics and Data Generation

The natural evaluation metric for the model-based clustering algorithm is the log-likelihood value. We take 10% random samples to examine the Gaussian mixture
models generated. To be more observable, we average the log-likelihood over the samples.

Since some distance-based clustering algorithms are involved, we also use the clustering accuracy to measure the accuracy if an original classification is given. For the synthetic data sets, we use the classification generated by the original mixture model. The clustering accuracy is defined to be the proportion of cases for which the most likely cluster $k$ (according to the mixture model) is the true one [65].

We generate two groups of data sets based on random mixture models. The first group contains 2 data sets and the second group contains the remaining 8 data sets. The first 2 mixture models are on a 2-dimensional space. The
Figure 5.2: The clustering features and cluster centers generated by BIRCH for the fourth synthetic data set. A dot indicates a clustering feature, while an ‘x’ denotes a cluster center.

means of the Gaussian components are located on grids and are from $[1,1], [2,1], \ldots, [l, p K_m]$; $l_p K_m$. The mixing proportion $p_k$ ranges in $\left[\frac{1}{100K}, \frac{3}{100K}\right]$, and so the cluster sizes may be very skew. The variance for each attribute falls in $[0.001,0.5]$. The first data set and its mixture model are illustrated in Figure 2.1 on Page 17. The second group of 8 data sets are generated according to random Gaussian mixture models. The main difference from the first two ones is their mean vector generation. A pair of mean vectors are generated together to ensure that their Euclidean distance is 1.0. In other words, this pair of clusters are very close and are not well separated. A typical data set is illustrated in Figure 5.1. A dot indicates a data item, and a ‘+’ and its associated dotted ellipse indicate a Gaussian distribution. The number of data items ranged from 60,000 to 480,000,
and the number of clusters ranges from 6 to 41. All of these parameters can be found in Table 5.1.

Since BIRCH is a distance-based clustering algorithm, it is not very suitable to handle our synthetic data sets as discussed in Section 1.2.2. Consider the fourth data set depicted in Figure 5.1, the cluster centers generated by BIRCH for the data set are illustrated in Figure 5.2. In this figure, a dot indicates a clustering feature, while an ‘x’ denotes a cluster center. The cluster centers are very different from those in the original mixture model as shown in Figure 5.1. For example, the bottom-right cluster is partitioned into two clusters. As a comparison, Figure 5.3 illustrates a typical mixture model generated by the gEMACF algorithm. In this
Table 5.1: The clustering accuracy of five clustering algorithms on 10 data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>N</th>
<th>D</th>
<th>K</th>
<th>gEMACF</th>
<th>gExEM</th>
<th>gEMAWS</th>
<th>iEM</th>
<th>sampiEM</th>
<th>IBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60000</td>
<td>2</td>
<td>6</td>
<td>0.962</td>
<td>0.934</td>
<td>0.942</td>
<td>0.960</td>
<td>0.958</td>
<td>0.836</td>
</tr>
<tr>
<td>2</td>
<td>480000</td>
<td>2</td>
<td>16</td>
<td>0.885</td>
<td>0.882</td>
<td>0.819</td>
<td>0.855</td>
<td>0.827</td>
<td>0.671</td>
</tr>
<tr>
<td>3</td>
<td>100000</td>
<td>2</td>
<td>9</td>
<td>0.785</td>
<td>0.676</td>
<td>0.782</td>
<td>0.748</td>
<td>0.745</td>
<td>0.553</td>
</tr>
<tr>
<td>4</td>
<td>100000</td>
<td>2</td>
<td>20</td>
<td>0.851</td>
<td>0.847</td>
<td>0.739</td>
<td>0.858</td>
<td>0.846</td>
<td>0.632</td>
</tr>
<tr>
<td>5</td>
<td>120000</td>
<td>2</td>
<td>31</td>
<td>0.893</td>
<td>0.854</td>
<td>0.819</td>
<td>0.905</td>
<td>0.875</td>
<td>0.697</td>
</tr>
<tr>
<td>6</td>
<td>120000</td>
<td>2</td>
<td>41</td>
<td>0.873</td>
<td>0.865</td>
<td>0.727</td>
<td>0.871</td>
<td>0.839</td>
<td>0.657</td>
</tr>
<tr>
<td>7</td>
<td>100000</td>
<td>3</td>
<td>10</td>
<td>0.911</td>
<td>0.917</td>
<td>0.854</td>
<td>0.892</td>
<td>0.770</td>
<td>0.676</td>
</tr>
<tr>
<td>8</td>
<td>100000</td>
<td>4</td>
<td>10</td>
<td>0.931</td>
<td>0.927</td>
<td>0.920</td>
<td>0.930</td>
<td>0.925</td>
<td>0.825</td>
</tr>
<tr>
<td>9</td>
<td>100000</td>
<td>5</td>
<td>10</td>
<td>0.868</td>
<td>0.816</td>
<td>0.820</td>
<td>0.856</td>
<td>0.843</td>
<td>0.741</td>
</tr>
<tr>
<td>10</td>
<td>100000</td>
<td>6</td>
<td>10</td>
<td>0.934</td>
<td>0.934</td>
<td>0.800</td>
<td>0.904</td>
<td>0.896</td>
<td>0.690</td>
</tr>
<tr>
<td>Average</td>
<td>0.889</td>
<td>0.865</td>
<td>0.822</td>
<td>0.878</td>
<td>0.853</td>
<td>0.698</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure, a ‘*’ indicates a clustering feature. The generated Gaussian component is indicated by an ‘o’ and an ellipse, while an original Gaussian component is represented by a ‘+’ and a dotted ellipse. It is clear that the generated mixture model is quite similar to the original one.

### 5.2.3 Clustering Accuracy

Table 5.1 illustrates the clustering accuracy of the six algorithms on ten data sets. In the table, the bold text denotes the best one in the corresponding data set. $N$, $D$ and $K$ indicate the number of data items, the dimensionality, and the number of clusters, respectively. The ten data sets are generated according to 10 random mixture models with the respective sets of the values of $N$, $D$, and $K$.

It can be observed from Table 5.1 that the gEMACF algorithm generates better clustering results than the other ones except for the fourth, the fifth and the seventh data sets, for which the gEMACF algorithm performs slightly worse. Figure 5.3 illustrates a typical mixture model generated by the gEMACF algorithm which describes the data set in Figure 5.1 well. For the tenth data set in a 6-dimensional space, the clustering accuracy of both the gEMACF and the gExEM algorithms is 93.4%. The clustering accuracy values of the gEMAWS algorithm, iEM and sampiEM are 80.0%, 90.4%, and 89.6% respectively.
gEMACF and the gExEM algorithms can generate more accurate results. The clustering accuracy of IBS is only 69.0%, which is much worse than the other five model-based clustering algorithms.

The average clustering accuracy values over the 10 data sets of the gEMACF, iEM, gExEM, sampiEM, and gEMAWS algorithms are 88.9%, 87.8%, 86.5%, 85.3% and 82.2%, respectively. Thus the gEMACF algorithm generates more accurate results. It is interesting to see that the gEMACF algorithm performs even slightly better than iEM (the classical EM algorithm). One possible explanation is that the smaller number of clustering features may cause smaller number of local maxima in the log-likelihood space. The gEMACF algorithm performs much better than sampiEM because sampling usually introduces some biases.

The average clustering accuracy for the distance-based clustering algorithm, IBS, is 69.8%, which is much worse than those of the model-based clustering algorithms. The main reason is that all Gaussian components involved are so skew that they are not suitable for the distance-based clustering algorithms.

### 5.2.4 Scalability

Figure 5.4 illustrates the execution times for the six clustering algorithms on 8 4-dimensional data sets. All data sets have 10 clusters. Their sizes increase exponentially from 6,250 to 800,000. The parameters for these eight data sets and the clustering accuracy for different algorithms can be found in Table 5.2. To clearly show the scalability, logarithm axes are used in the figure.

It can be observed from Figure 5.4 that the execution times of the gEMACF algorithm increase very slowly with the number of data items. It takes 211.6 seconds for the data set with 6,250 data items, and takes 1,344.3 seconds for the data set with 800,000 data items. The execution times of the gExEM, gEMAWS and IBS algorithm increase as slowly as those of the gEMACF algorithm. These 4 algorithms all scale up well with large data sets. However, as listed in Table
Figure 5.4: The execution times on eight 4-dimensional data sets.

5.2, the gEMACF algorithm normally generates more accurate mixture models than the other three algorithms. The average clustering accuracy values of the gEMACF, gExEM, gEMAWS and IBS algorithms are 91.9%, 88.5%, 83.2% and 78.5%, respectively. The gEMACF algorithm at least makes a 3.4% improvement over its three counterparts.

The execution times of sampiEM range from 30.9 seconds to 4050.0 seconds. The execution times increase linearly with the data size, because sampiEM has no restriction on the amount of the main memory used. The gEMACF algorithm needs longer execution time for the first three small data sets mainly because the data summarization procedure takes longer than the random sampling procedure in sampiEM. However, the data summarization overhead becomes relatively small as the data size increases. In fact, the execution time ratios of the gEMACF algorithm to sampiEM are 2.5 and 3.0 for the data sets with 400,000 and 800,000 data items, respectively. In addition, the clustering accuracy of sampiEM is much
Table 5.2: The clustering accuracy on eight 4-dimensional data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>N</th>
<th>D</th>
<th>K</th>
<th>gEMACF</th>
<th>gExEM</th>
<th>gEMAWS</th>
<th>iEM</th>
<th>sampiEM</th>
<th>IBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDSA</td>
<td>6250</td>
<td>4</td>
<td>10</td>
<td>0.948</td>
<td>0.885</td>
<td>0.858</td>
<td>0.958</td>
<td>0.874</td>
<td>0.822</td>
</tr>
<tr>
<td>iDSB</td>
<td>12500</td>
<td>4</td>
<td>10</td>
<td>0.931</td>
<td>0.931</td>
<td>0.795</td>
<td>0.923</td>
<td>0.884</td>
<td>0.739</td>
</tr>
<tr>
<td>iDSC</td>
<td>25000</td>
<td>4</td>
<td>10</td>
<td>0.947</td>
<td>0.885</td>
<td>0.799</td>
<td>0.929</td>
<td>0.882</td>
<td>0.724</td>
</tr>
<tr>
<td>iDSD</td>
<td>50000</td>
<td>4</td>
<td>10</td>
<td>0.932</td>
<td>0.916</td>
<td>0.879</td>
<td>0.918</td>
<td>0.863</td>
<td>0.821</td>
</tr>
<tr>
<td>iDSE</td>
<td>100000</td>
<td>4</td>
<td>10</td>
<td>0.933</td>
<td>0.927</td>
<td>0.920</td>
<td>0.930</td>
<td>0.925</td>
<td>0.825</td>
</tr>
<tr>
<td>iDSF</td>
<td>200000</td>
<td>4</td>
<td>10</td>
<td>0.901</td>
<td>0.838</td>
<td>0.828</td>
<td>0.905</td>
<td>0.871</td>
<td>0.723</td>
</tr>
<tr>
<td>iDSG</td>
<td>400000</td>
<td>4</td>
<td>10</td>
<td>0.866</td>
<td>0.819</td>
<td>0.811</td>
<td>0.870</td>
<td>0.832</td>
<td>0.856</td>
</tr>
<tr>
<td>iDSH</td>
<td>800000</td>
<td>4</td>
<td>10</td>
<td>0.938</td>
<td>0.881</td>
<td>0.763</td>
<td>0.951</td>
<td>0.905</td>
<td>0.772</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>0.919</td>
<td>0.885</td>
<td>0.832</td>
<td>0.923</td>
<td>0.879</td>
<td>0.785</td>
</tr>
</tbody>
</table>

worse than the gEMACF algorithm. As listed in Table 5.2, the average clustering accuracy of the gEMACF algorithm is 4.0% higher than the value of 87.9% for sampiEM.

The execution times of iEM increase from 512.0 seconds to 307,654.6 seconds for the data set with 800,000 data items. The execution time of iEM increases almost linearly with the data size because the amount of the main memory used is not restricted during its execution. For the data set with 800,000 data items, the gEMACF algorithm runs 228.9 times faster than iEM. For the first data set with 6,250 data items, the speedup factor is about 2.4. For other data sets, the speedup factor of the gEMACF algorithm to iEM ranges from 5.5 to 142.7. Thus, the gEMACF algorithm can run one or two orders of magnitude faster than the iEM algorithm. In addition, it can be observed from the clustering accuracy values listed in Table 5.2, the gEMACF algorithm sometimes generates more accurate results than iEM. The clustering accuracy of the gEMACF algorithm is 91.9% which is comparable with the value of 92.3% for iEM.

5.2.5 Sensitivity

The influences of the data summarization procedure on the performance of EMACF is discussed here. The data set shown in Figure 2.1 is taken as an example. Figure 5.5 illustrates the clustering accuracy of five clustering algorithms for different
Figure 5.5: Sensitivity of clustering algorithms to the data summarization or sampling procedures.

data summarization procedures. The data summarization procedures are manipulated by varying grid structures. For the first grid structure, we partition two attributes into 40 segments and get 1,600 cells. The grid structure is indicated by 40*40 in Figure 5.5. The sampiEM(M) algorithm refers to that the EM algorithm works on \( M \) samples, say, 1,600 for the first grid structure.

For the first five grid structures, the segment numbers for each attribute are 40, 32, 24, 16 and 8 respectively. That is, the cell granularity increases gradually. As shown in Figure 5.5, the clustering accuracy of all five algorithms decreases gradually. The clustering accuracy of the gEMAWS algorithm decreases dramatically. However, the clustering accuracy of the gEMACF algorithm decreases very slowly. The accuracy ranges from 97.2% to 90.5%. Even for the 8*8 grid structure, the gEMACF algorithm can generate good clusters with clustering accuracy of 90.5%. A typical mixture model is illustrated by six solid ellipses and their associated ‘o’ in Figure 5.6. These six dotted ellipses and their associated ‘+’ indicate the original mixture model. Both mixture models are quite close to each
Figure 5.6: A typical mixture model generated by the gEMACF algorithm from 8×8 grid structure. The 64 ‘*’ indicate the 64 clustering features. A ellipse and its associated ‘o’ indicate a Gaussian component generated, and a dotted ellipse and its associated ‘+’ indicate an original Gaussian component.

The last four grid structures aim to generate very skew subclusters. For example, the 8×128 grid structure divides the two attributes into 8 and 128 segments respectively. So the cell width is 16 times of the cell height as the small cells shown in Figure 5.7. The clustering accuracy values of the gEMACF, gExEM, gEMAWS and IBS algorithms decrease gradually when the cells become elongated. However, the gEMACF algorithm degrades most slowly. Its clustering accuracy ranges from 99.8% and 89.2%, which is much better than the other algorithms. One typical mixture model generated by the gEMACF algorithm from the 8×128 grid structure is illustrated in Figure 5.7. The generated mixture model indicated by the ellipses are close to the original mixture model indicated by the dotted ellipses.
Figure 5.7: A typical mixture model generated by the gEMACF algorithm from 8*128 grid structure. The 748 '*' indicate the 748 clustering features. A ellipse and its associated ‘o’ indicate a Gaussian component generated, and a dotted ellipse and its associated ‘+’ indicate an original Gaussian component.

In summary, the gEMACF algorithm is least sensitive to the data summarization procedures. The gExEM algorithm is the second. The gEMAWS algorithm is most sensitive to the data summarization procedures whose clustering accuracy values vary greatly from 94.6% to 60.9%.

5.3 Performance of bEMACF

5.3.1 Methodology

When EMACF works on the clustering features generated by the BIRCH’s data summarization procedure, we may have another scalable model-based clustering system. We call it the bEMACF algorithm. We compare the bEMACF algorithm with four different clustering algorithms as follows.
The classical EM algorithm: It is denoted by iEM below. Here we use ‘i’ to denote that attributes are statistically independent.

The sampling EM algorithm: It is iEM working on 5% random samples. It is denoted by sampiEM below.

The bExEM algorithm: The ExEM algorithm works on the clustering features generated by the BIRCH's data summarization procedure. It can run faster than the SEM algorithm in [11].

The gEMACF algorithm: It is EMACF works on the clustering features generated by our adaptive grid-based data summarization procedure. We have studies it comprehensively in the previous section. For convenient comparison, we list its results again.

When we use the data summarization procedure of BIRCH to generate the CF-tree and clustering features, we use the default parameter setting except for the following three parameters. The maximum main memory used is 8,192,000 bytes, the outliers are defined as the leaf entries containing less than 10% of the average number of data items per leaf entry, and the input range is 4,000. In other words, the maximum number of clustering features is 4,000. This number equals the maximum number of clustering features generated by the grid-based data summarization. All these algorithms are coded in MATLAB and experiments are conducted on a Sun Enterprise E4500 server. These model-based clustering algorithms are initialized with the cluster centers generated by the K-means algorithm. We use two termination criteria for all algorithms. One is the maximal number of iterations, 500. The clustering algorithms is also terminated if the successive log-likelihood change is within $10^{-5}$ of current value as used in [65, 83]. In other words, $\epsilon$ is set to $10^{-5}$ in iEM, ExEM and EMACF. All experimental results reported are averaged on 10 independent runs.
Table 5.3: The clustering accuracy of 5 clustering algorithms on 10 data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>N</th>
<th>D</th>
<th>K</th>
<th>gEMACF</th>
<th>bEMACF</th>
<th>bExEM</th>
<th>iEM</th>
<th>sampiEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60000</td>
<td>2</td>
<td>6</td>
<td>0.962</td>
<td>0.954</td>
<td>0.942</td>
<td>0.960</td>
<td>0.958</td>
</tr>
<tr>
<td>2</td>
<td>480000</td>
<td>2</td>
<td>16</td>
<td>0.885</td>
<td>0.882</td>
<td>0.877</td>
<td>0.855</td>
<td>0.827</td>
</tr>
<tr>
<td>3</td>
<td>1000000</td>
<td>2</td>
<td>9</td>
<td>0.785</td>
<td>0.781</td>
<td>0.731</td>
<td>0.748</td>
<td>0.745</td>
</tr>
<tr>
<td>4</td>
<td>100000</td>
<td>2</td>
<td>20</td>
<td>0.851</td>
<td>0.859</td>
<td>0.848</td>
<td>0.858</td>
<td>0.846</td>
</tr>
<tr>
<td>5</td>
<td>120000</td>
<td>2</td>
<td>31</td>
<td>0.893</td>
<td>0.923</td>
<td>0.881</td>
<td>0.905</td>
<td>0.875</td>
</tr>
<tr>
<td>6</td>
<td>120000</td>
<td>2</td>
<td>41</td>
<td>0.873</td>
<td>0.877</td>
<td>0.830</td>
<td>0.871</td>
<td>0.839</td>
</tr>
<tr>
<td>7</td>
<td>1000000</td>
<td>3</td>
<td>10</td>
<td>0.911</td>
<td>0.844</td>
<td>0.805</td>
<td>0.892</td>
<td>0.770</td>
</tr>
<tr>
<td>8</td>
<td>1000000</td>
<td>4</td>
<td>10</td>
<td>0.931</td>
<td>0.935</td>
<td>0.937</td>
<td>0.930</td>
<td>0.925</td>
</tr>
<tr>
<td>9</td>
<td>1000000</td>
<td>5</td>
<td>10</td>
<td>0.868</td>
<td>0.932</td>
<td>0.922</td>
<td>0.856</td>
<td>0.843</td>
</tr>
<tr>
<td>10</td>
<td>1000000</td>
<td>6</td>
<td>10</td>
<td>0.934</td>
<td>0.941</td>
<td>0.938</td>
<td>0.904</td>
<td>0.896</td>
</tr>
<tr>
<td>Average</td>
<td>0.889</td>
<td>0.893</td>
<td>0.871</td>
<td>0.878</td>
<td>0.853</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.3.2 Clustering Accuracy

Table 5.3 lists the clustering accuracy of the five algorithms for 10 synthetic data sets. In the table, the bold text denotes the best one for the corresponding data set. $N$, $D$ and $K$ indicate the number of data items, the dimensionality, and the number of clusters, respectively. The data size ranges from 60,000 to 480,000, the data dimensionality ranges from 2 to 6, and the number of clusters varies from 6 to 41.

Let us take a close look at the performance of the five algorithms on the fourth data set shown in Figure 5.1. The grid-based data summarization procedure generates 656 clustering features. The average accuracy of the mixture models generated by the gEMACF algorithm from the clustering features is 85.1%. This value falls between the clustering accuracy values of iEM and sampiEM. The clustering features and a typical mixture model can be found in Figure 5.3. The BIRCH’s data summarization procedure generates 1,686 clustering features. The mixture models generated by the bEMACF algorithm from these clustering features describe the data set well. The clustering accuracy of the bEMACF algorithm is 85.9%, which is slightly better than the value of 84.8% for iEM. And the bEMACF algorithm is the best one among the five algorithms. The clustering features and a typical mixture model generated by the bEMACF algorithm can be found in Figure 5.8. It is clear that the bEMACF algorithm can generate
Figure 5.8: A typical mixture model generated by the bEMACF algorithm for the fourth data set. A dot indicates a clustering feature. An ‘o’ and an ellipse indicate a Gaussian component generated, while a ‘+’ and a dotted ellipse indicate an original Gaussian component.

mixture models very close to the original one.

The bEMACF algorithm usually performs better than the classical EM algorithm — iEM. Among the 10 data sets, the clustering accuracy of the bEMACF algorithm is higher than that of iEM on 8 data sets. The average clustering accuracy of the bEMACF algorithm is 89.3%. The clustering accuracy of iEM is 87.8%, which is about 1.5% worse than that of the bEMACF algorithm. This may be caused by two factors. The first one is that the BIRCH’s data summarization procedure can capture the clustering structure very well. The other one is that the smaller number of clustering features may cause smaller number of local maxima in the log-likelihood space.
Figure 5.9: Execution times of five clustering algorithms for eight 4-dimensional data sets.

The clustering results of sampiEM are not as good as those of other clustering algorithms. The average clustering accuracy of sampiEM is 85.3%, which is the lowest among the five algorithms. The main reason is that 5% random samples may introduce some biases. The average clustering accuracy value of the bExEM algorithm is 87.1%, which is 2.2% lower than the value of 89.3% for the bEMACF algorithm.

As shown in Table 5.3, the gEMACF algorithm sometimes outperforms the bEMACF algorithm, and vice versa. Especially, the gEMACF algorithm performs better than the bEMACF algorithm on the data sets with a small number of clusters in a low-dimensional space. The average clustering accuracy values of the gEMACF and bEMACF algorithms, respectively, are 88.9% and 89.3%. Hence, the gEMACF algorithm is comparable with the bEMACF algorithm.
5.3.3 Scalability

Figure 5.9 illustrates the execution times for the five clustering algorithms on eight 4-dimensional data sets. All data sets have 10 clusters. The difference of these data sets is only their sizes. Their data sizes increase exponentially from 6,250 to 800,000. The parameters of these 8 data sets and the clustering accuracy for different algorithms can be found in Table 5.4. To clearly show the scalability, logarithm axes are used in the figure.

It can be observed from Figure 5.9 that the execution times of the bEMACF algorithm increase very slowly with the number of data items. The bEMACF algorithm takes 197.4 seconds for the data set with 6,250 data items, and takes 611.0 seconds for the data set with 800,000 data items. The execution times of the bExEM algorithm increase as slowly as those of the bEMACF algorithm. Both the bEMACF and bExEM algorithms run slightly faster than the gEMACF algorithm. This is mainly caused by their different data summarization procedures. For example, for the largest data set with 800,000 data items, the BIRCH’s data summarization procedure takes 216.3 seconds to generate 2,763 clustering features. The adaptive grid-based data summarization needs 985.4 seconds to generate 1,739 clustering features. These three algorithms all scale up well with large data sets. However, as listed in Table 5.4, the average clustering accuracy values of the bEMACF, the gEMACF and the bExEM algorithms are 93.0%, 91.9% and 91.8%, respectively. The bEMACF algorithm normally generates more accurate mixture models than the other two algorithms.

The execution times of sampiEM range from 30.9 seconds to 4050.0 seconds. The execution times increase linearly with the data size, since sampiEM has no restriction on the amount of the main memory used. The bEMACF algorithm needs longer execution time for the first three small data sets, because the data summarization procedure takes longer execution time than the random sampling
Table 5.4: Clustering accuracy of five clustering algorithms for eight 4-dimensional data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>N(1000)</th>
<th>D</th>
<th>K</th>
<th>gEMACF</th>
<th>bEMACF</th>
<th>bExEM</th>
<th>iEM</th>
<th>sampiEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDSA</td>
<td>6.25</td>
<td>4</td>
<td>10</td>
<td>0.948</td>
<td>0.903</td>
<td>0.873</td>
<td>0.958</td>
<td>0.874</td>
</tr>
<tr>
<td>iDSB</td>
<td>12.5</td>
<td>4</td>
<td>10</td>
<td>0.931</td>
<td>0.894</td>
<td>0.917</td>
<td>0.923</td>
<td>0.884</td>
</tr>
<tr>
<td>iDSC</td>
<td>25</td>
<td>4</td>
<td>10</td>
<td>0.947</td>
<td>0.888</td>
<td>0.885</td>
<td>0.929</td>
<td>0.882</td>
</tr>
<tr>
<td>iDSD</td>
<td>50</td>
<td>4</td>
<td>10</td>
<td>0.992</td>
<td>0.953</td>
<td>0.947</td>
<td>0.918</td>
<td>0.863</td>
</tr>
<tr>
<td>iDSE</td>
<td>100</td>
<td>4</td>
<td>10</td>
<td>0.933</td>
<td>0.935</td>
<td>0.937</td>
<td>0.930</td>
<td>0.925</td>
</tr>
<tr>
<td>iDSF</td>
<td>200</td>
<td>4</td>
<td>10</td>
<td>0.901</td>
<td>0.947</td>
<td>0.901</td>
<td>0.905</td>
<td>0.871</td>
</tr>
<tr>
<td>iDSG</td>
<td>400</td>
<td>4</td>
<td>10</td>
<td>0.866</td>
<td>0.970</td>
<td>0.935</td>
<td>0.870</td>
<td>0.832</td>
</tr>
<tr>
<td>iDSH</td>
<td>800</td>
<td>4</td>
<td>10</td>
<td>0.938</td>
<td>0.952</td>
<td>0.952</td>
<td>0.951</td>
<td>0.905</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>0.919</td>
<td>0.930</td>
<td>0.918</td>
<td>0.923</td>
<td>0.879</td>
</tr>
</tbody>
</table>

procedure in sampiEM. However, the data summarization overhead becomes relatively small as the data size increases. In fact, the execution time ratios of the bEMACF algorithm to sampiEM are, respectively, 4.5 and 4.3 for the data sets with 400,000 and 800,000 data items. In addition, the average clustering accuracy of sampiEM is much worse than that of the bEMACF algorithm. As listed in Table 5.4, the average clustering accuracy of the bEMACF algorithm is 5.1% higher than the value of 87.9% for sampiEM.

The execution times of iEM increase from 512.0 seconds to 307,654.6 seconds for the data set with 800,000 data items. The execution times of iEM increase almost linearly with the data size. It is because that the amount of the main memory used is not restricted during the execution. For the data set with 800,000 data items, the bEMACF algorithm runs 503.5 times faster than iEM. For the first data set with 6,250 data items, the speedup factor is about 2.6. For the other data sets, the speedup factors of the bEMACF algorithm to iEM range from 4.4 to 246.1. Thus, the bEMACF algorithm can run one or two orders of magnitude faster than the iEM algorithm. In addition, it can be observed from the clustering accuracy values listed in Table 5.4 that the bEMACF algorithm usually generates more accurate clustering results than iEM. The average clustering accuracy of the bEMACF algorithm is 93.0% which is slightly better than the value of 92.3% of iEM.
5.3.4 Application to Real World Data Sets

For the real world data sets, since we don’t know the original classification, we cannot employ the clustering accuracy to measure the Gaussian mixture models generated by clustering algorithms. We use the average log-likelihood to measure the accuracy of the mixture models generated. The likelihood is the joint probability density for all the data items. It measures to what degree the mixture model matches the data set. Thus, the larger the average log-likelihood is, the better the match between the mixture model and the data set is. Here we only list the experimental results of the bEMACF, bExEM, iEM and sampiEM algorithms.

The first data set, the Forest CoverType Data, is downloaded from the UCI Knowledge Discovery in Databases Archive (http://kdd.ics.uci.edu). It describes forest cover for 30x30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The data set has 581,012 data items. We use five quantitative attributes including Elevation, Aspect, Horizontal Distance To Hydrology, Horizontal Distance To Roadways, and Horizontal Distance To Fire Points. All attribute values are scaled into [0, 3]. We use a Gaussian mixture model with 15 components (clusters) to describe the data set. The BIRCH’s data summarization procedure generates 3,186 clustering features.

Table 5.5 lists the experimental results. The standard deviation of the log-likelihood is also included. Here sampiEM(15%) represents the classical EM algorithm working on 15% random samples and sampiEM(5%) represents the classical EM algorithm working on 5% random samples. We used sampiEM(15%) instead of iEM since iEM cannot generate a mixture model after running for 100 hours. In fact, even sampiEM(15%) takes 53,475.2 seconds and sampiEM(5%) takes 17,872.4 seconds.

On average, the bEMACF algorithm takes about 2,064.9 seconds. It runs 25.9 times faster than sampiEM(15%), and 7.50 times faster than sampiEM(5%).
Table 5.5: The performance of four clustering algorithms on the Forest CoverType Data.

<table>
<thead>
<tr>
<th></th>
<th>bEMACF</th>
<th>bExEM</th>
<th>sampiEM(15%)</th>
<th>sampiEM(5%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>-3.242 ± 0.019</td>
<td>-3.256 ± 0.019</td>
<td>-3.250 ± 0.018</td>
<td>-3.252 ± 0.022</td>
</tr>
<tr>
<td>Time(Sec.)</td>
<td>2064.9 ± 891.2</td>
<td>1689.4 ± 618.9</td>
<td>53475.2 ± 10166.3</td>
<td>17872.4 ± 5198.4</td>
</tr>
</tbody>
</table>

Moreover, the log-likelihood value of the bEMACF algorithm is -3.242. The log-likelihood values of sampiEM(15%) and sampiEM(5%) are -3.250 and -3.252, respectively. So, the bEMACF algorithm can generate more accurate mixture models than both sampiEM(15%) and sampiEM(5%). The difference of log-likelihood between sampiEM(15%) and the bEMACF algorithm is 0.008. The difference is quite interesting if we note that the standard deviation of the log-likelihood is around 0.018. Though the bExEM algorithm runs slightly faster than the bEMACF algorithm, it generates the worst mixture models with the log-likelihood value of -3.256.

The second data set, the Census-Income Database, is also downloaded from the UCI KDD archive. This data set contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. The data set contains 299,285 data items. We used 3 continuous attributes including Age, Dividends From Stocks, and Weeks Worked In Year. All attribute values are scaled into [0, 3]. The mixture model used has 10 Gaussian distribution components. The BIRCH’s data summarization procedure generates 3,836 clustering features.

Table 5.6 lists the performance of the four algorithms for the Census-Income Database. The log-likelihood of the bEMACF algorithm is -0.747, which is the best among the four algorithms. It is interesting to see that the log-likelihood of iEM is also -0.747. Thus the models generated by the bEMACF algorithm and iEM are compatible. However, the two algorithms take significantly different execution times. On average, the bEMACF algorithm takes about 1,361.3 seconds
Table 5.6: The performance of four clustering algorithms on the Census-Income Database.

<table>
<thead>
<tr>
<th>Census</th>
<th>bEMACF</th>
<th>bExEM</th>
<th>iEM</th>
<th>sampiEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>-0.747 ± 0.003</td>
<td>-0.750 ± 0.005</td>
<td>-0.747 ± 0.008</td>
<td>-0.750 ± 0.007</td>
</tr>
<tr>
<td>Time(Sec.)</td>
<td>1361.3 ± 371.6</td>
<td>1205.3 ± 362.2</td>
<td>209865.6 ± 35892.2</td>
<td>4300.7 ± 1996.4</td>
</tr>
</tbody>
</table>

while iEM takes 172,181.6 seconds. The ratio of the execution times is 126.5. In other words, the bEMACF algorithm is two orders of magnitude faster than iEM without loss of accuracy.

The log-likelihood of sampiEM is -0.750. Note that the standard deviation of the log-likelihood of the bEMACF algorithm is 0.003. So, the mixture models generated by the bEMACF algorithm is scientifically better than sampiEM. In addition, sampiEM needs 4300.7 seconds for execution which is 3.2 times longer than the bEMACF algorithm. Thus, the bEMACF algorithm outperform sampiEM in terms of both the execution time and the clustering accuracy.

The bExEM algorithm runs slightly faster than the bEMACF algorithm. However, the log-likelihood of the bExEM algorithm is -0.750, which is lower than the value of -0.747 for the bEMACF algorithm.

The third data set, the California housing data, describes all the block groups in California from the 1990 USA Census. It is downloaded from www.spatial-statistics.com. It has 20,640 data items. In our experiments, we use 8 numeric attributes including Median House Value, Median Income, Total Rooms, Total Bedrooms, Population, Households, Latitude, and Longitude. All of the 8 attributes are scaled into [0, 3]. We use a mixture model of seven Gaussian components to describe the data set. The data summarization procedure of BIRCH generates 2,907 clustering features.

Table 5.7 lists the performance of the four clustering algorithms for the data set. The average log-likelihood of the bEMACF algorithm is 3.512. It is better than 3.462, the log-likelihood value of the bExEM algorithm. On average, the
Table 5.7: The performance of four clustering algorithms on the California housing data.

<table>
<thead>
<tr>
<th>Housing</th>
<th>bEMACF</th>
<th>bExEM</th>
<th>iEM</th>
<th>sampiEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>3.512 ± 0.141</td>
<td>3.462 ± 0.166</td>
<td><strong>3.804 ± 0.162</strong></td>
<td>3.380 ± 0.281</td>
</tr>
<tr>
<td>Time(Sec.)</td>
<td>684.9 ± 113.2</td>
<td>719.9 ± 468.1</td>
<td>4452.6 ± 691.40</td>
<td>263.8 ± 59.8</td>
</tr>
</tbody>
</table>

bEMACF algorithm takes 684.9 seconds while the bExEM algorithm takes 719.9 seconds which is slightly longer.

The performance of the bEMACF algorithm lies between iEM and sampiEM. The log-likelihood value of sampiEM is 3.380, which is smaller than the value of 3.512 for the bEMACF algorithm. The sampiEM takes about 263.8 seconds, and thus it is faster than the bEMACF algorithm. The log-likelihood of iEM is 3.804, which is better than that of the bEMACF algorithm. However, iEM takes 4,452.6 seconds to execute. The bEMACF algorithm runs 6.5 times faster than the iEM algorithm. For this data set, the advantage of the bEMACF algorithm is not so apparent partially because the data size is not very large.

So, for large real world data sets, the bEMACF algorithm can generate mixture models one or two order magnitude faster than the classical EM algorithm with little loss of accuracy. It can generate better mixture models than the sampling EM algorithm in terms of both time and accuracy. It may generate more accurate mixture models than the bExEM algorithm with similar execution time.

5.4 Chapter Summary

In this chapter, we have established a new iterative algorithm EMACF (the EM Algorithm for Clustering Features). It is the first mathematically sound algorithm to generate mixture models from clustering features directly. It embeds the whole clustering features into both its E-step and M-step in order to generate accurate mixture models. Its computational complexity is linear with the number
of clustering features. We have proved that EMACF converges to local maxima.

We have established two scalable clustering systems: the gEMACF and bEMACF algorithms, by combining EMACF with the adaptive grid-based and the BIRCH’s data summarization procedures. The comprehensive experimental results on both the synthetic and real world data sets have substantiated several interesting features of our proposed algorithms. EMACF is less sensitive to the data summarization procedures than the ExEM algorithm and the sampling EM algorithm. The scalable clustering systems based on EMACF, the gEMACF and the bEMACF algorithms, can generate more accurate clustering results than the gExEM and the bExEM algorithms, which are based on the ExEM algorithm. Both the gEMACF and bEMACF algorithms are scalable with large data sets. Furthermore, they can run one or two orders of magnitude faster than the classical EM algorithm with no or little loss of accuracy. They can generate more accurate clustering results and run faster than the sampling EM algorithm for large data sets.
Chapter 6

Scalable Cluster Analysis by Working on Data Summaries

It is clear that the correlations among data attributes are ubiquitous. For example, it is obvious to see the correlation between the two attributes, Latitude and Longitude, in the California housing data as shown in Figure 6.1. Most of clustering algorithms, explicitly or implicitly, suppose that data attributes are statistically independent. When they handle a data set as in Figure 6.1, the clustering results are sometimes far away from the inherent structures within the data set. Figure 6.2 illustrates a typical mixture model generated by EMACF from 783 clustering features. An ‘o’ and its associated ellipse indicate, respectively, a center and a contour of a Gaussian distribution component. Each ‘*’ represents a clustering feature. The mixture model cannot reveal the inherent clusters very well.

This chapter aims to establish some scalable model-based clustering algorithms that take account of the correlations among attributes explicitly. Figure 6.3 illustrates a typical mixture model generated by an algorithm proposed in this chapter. In this figure, a ‘*’ represents a data summary. The mixture model has seven Gaussian components. The Gaussian component, indicated by the biggest ellipse, describes the noisy data, and the other six Gaussian components describe six clusters. The cluster structure in the data set shown in Figure 6.1 is clearly
In this chapter, we first design a simplified data summary representation scheme with a good mathematical foundation. Then we propose a new density function for the simplified data summaries. After that, EMADS (the EM Algorithm for Data Summaries) is derived based on the principle of the general EM algorithm. We then prove that EMADS converges to a local maximum definitely. After that, EMADS is employed to establish scalable clustering systems in Section 6.2. Combining with the grid-based data summarization and the BIRCH’s data summarization procedures, we can establish two clustering systems, the gEMACF and bEMADS algorithms. The sensitivity, the clustering accuracy, and the scalability of EMADS are examined. We finally apply the proposed bEMADS to three
Figure 6.2: A typical mixture model generated by EMACF from 783 clustering features for the 2-D California housing data. A Gaussian distribution component is indicated by an ‘o’ and its associated ellipse. Each ‘*’ represents a clustering feature.

real world data sets.

6.1 EMADS

Our in-memory clustering algorithm is expected to generate mixture models from the summary statistics of subclusters effectively. To better capture the correlations among attributions, we have to retain the correlations among attributes within subclusters. That is why we use a data summary $DS_m = \{n_m, \nu_m, \Gamma_m\} (m = 1, \ldots, M)$ where $M$ is the number of subclusters) to represent a subcluster. As defined in Chapter 4, the data summary contains the zeroth, first, and second
moments of a subcluster of data items. We may derive the covariance matrix easily from the data summary.

6.1.1 A Simplified Data Summary Representation Scheme

Similar to the derivation of EMACF (the EM Algorithm for Clustering Features) in Chapter 5, we should embed the data distribution information of subclusters into both the density function and the M-step of our EMADS algorithm. The first problem is how to effectively incorporate matrix $\Gamma_m$ into a Gaussian density
function with parameters $\theta_k = \{\mu_k, \Sigma_k\}$,
\[
\phi(\nu_m|\theta_k) = \exp \left\{ -\frac{1}{2} (\nu_m - \mu_k)^T \Sigma_k^{-1} (\nu_m - \mu_k) \right\}.
\]
(6.1)
In other words, the problem is how to associate matrix $\Gamma_m$ with a numerical value. One possible answer is in a form similar to the expression $(\nu_m - \mu_k)^T \Sigma_k^{-1} (\nu_m - \mu_k)$.

For the M-step, we may get some inspiration from the approximation of the M-step of the classical EM algorithm for Gaussian mixture models. We make this approximation because the data items within a subcluster have similar behaviors in both the E-step and the M-step. In other words, they have similar membership probability $t_{ik}$. We rewrite Eq.(2.7) by putting data items within a subcluster together
\[
N_p^{(j+1)} \cdot \Sigma_k^{(j+1)} = \sum_{i=1}^{N} \sum_{r_{ik}} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^T
\]
\[
= \sum_{m=1}^{M} \sum_{x_i \in \text{the } m^{th} \text{ subcluster}} t_{ik}^{(j)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^T.
\]
For simplification, we use $x_i \in DS_m$ to indicate that a data item $x_i$ belongs to the $m^{th}$ subcluster below. If data items within the $m^{th}$ subcluster have the same membership probability $r_{mk}^{(j)}$, we get
\[
N_p^{(j+1)} \Sigma_k^{(j+1)} = \sum_{m=1}^{M} r_{mk}^{(j)} \sum_{x_i \in DS_m} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^T
\]
\[
= \sum_{m=1}^{M} r_{mk}^{(j)} n_m \left[ \sum_{x_i \in DS_m} \frac{x_i x_i^T}{n_m} - \left( \sum_{x_i \in DS_m} \frac{x_i}{n_m} \right) \left( \sum_{x_i \in DS_m} \frac{x_i}{n_m} \right)^T \right]
\]
\[
+ \left( \sum_{x_i \in DS_m} \frac{x_i}{n_m} - \mu_k^{(j+1)} \right) \left( \mu_k^{(j+1)} \right)^T \right]
\]
(6.2)
If we approximate $(\Gamma_m - \nu_m \nu_m^T)$ with the cross (outer) product of a vector, say, $(\Gamma_m - \nu_m \nu_m^T) \approx \delta_m \delta_m^T$, then the vector $\delta_m$ can be treated in a similar way as
\( (\nu_m - \mu_k^{(j+1)}) \) in Eqs.(6.1) and (6.2). In other words, similar to the item \( (\nu_m - \mu_k)^T \Sigma_k^{-1} (\nu_m - \mu_k) \) in Gaussian density function, we can insert an item \(-\frac{1}{2} \delta_m^T \Sigma_k^{-1} \delta_m\) into our new density function. Finally, the data distribution within a subcluster may be reflected in our density function.

Now we design a vector \( \delta_m \) to best approximate the matrix \( (\Gamma_m - \nu_m \nu_m^T) \). First we note that \( (\Gamma_m - \nu_m \nu_m^T) \) is a positive semi-definite matrix. In order to keep as much information as possible, the vector should be on the same direction as the principal component, \( c_1 \) of the matrix [92]. Associated with the component vectors, we introduce a new definition.

**Definition 6.1** The \( d^{th} \) covariance vector, corresponding to the \( d^{th} \) eigenvalue \( \lambda_d \), is defined as \( \sqrt{\lambda_d} c_d \), where \( c_d \) is the component vector corresponding to the \( d^{th} \) eigenvalue.

The following theorem says that the cross product of the first covariance vector can approximate a positive semi-definite matrix most closely.

**Theorem 6.2**  For any positive semi-definite matrix \( \Gamma \in \mathbb{R}^{D \times D} \), its first covariance vector \( \delta = \arg \min_y \| \Gamma - yy^T \| \) where \( \| \Gamma \| = \left( \frac{1}{2} \sum_{i,j=1}^{D} \gamma_{ij}^2 \right)^{\frac{1}{2}} \).

**Proof:** First, we prove that \( f(y) = \| \Gamma - yy^T \|^2 \) reaches local minima when \( y \) are covariance vectors. We set its partial derivative to zero, and get

\[
\frac{\partial f(y)}{\partial y_d} = \frac{\partial}{\partial y_d} \left( \sum_{i,j=1}^{D} (\gamma_{ij} - y_i y_j)^2 \right) = 4 \sum_{i=1}^{D} [(\gamma_{d1} - y_d y_i) y_i] = 0.
\]

Combining equations for all \( d \), we have \( \Gamma y = \| y \|^2 y \). This indicates that \( y \) is an eigenvector of \( \Gamma \) and its norm equals the square root of the associated eigenvalue. In other words, \( y \) is a covariance vector.

Now let us prove the first covariance vector reaches the minimum of \( \| \Gamma - yy^T \| \).

As a symmetric matrix, \( \Gamma \) has an orthonormal basis consisting of component vectors, \( C = \{ c_1, c_2, \cdots, c_D \} \) [55]. We have a decomposition, \( \Gamma = C \Lambda C^T \), where \( \Lambda \)
is the diagonal matrix consisting of the corresponding eigenvalue \( \lambda_d \). For convenience, these eigenvalues are sorted in a descending way. We have \( \Gamma = \sum_{i=1}^{D} \lambda_i c_i^T c_i \).

So, \( \sqrt{\lambda_i} c_i \) is the \( i^{th} \) covariance vector. Then we have

\[
||\Gamma - \lambda_d c_d c_d^T||^2 = \left( \sum_{i \neq d} \lambda_i c_i^T c_i \right)^2 = \sum_{k,l} \left[ \sum_{i \neq d} \lambda_i c_{ki} c_{li} \right]^2
\]

\[
= \sum_{k,l} \left[ \sum_{i \neq d} \lambda_i^2 c_{ki}^2 c_{li}^2 + 2 \sum_{i \neq d, j \neq d, i \neq j} \lambda_i \lambda_j c_{ki} c_{li} c_{kj} c_{lj} \right]
\]

\[
= \sum_{i \neq d} \lambda_i^2 + 2 \sum_{i \neq d, j \neq d, i \neq j} \sum_{k,l} \lambda_i \lambda_j c_{ki} c_{li} c_{kj} c_{lj}
\]

\[
= \sum_{i \neq d} \lambda_i^2. \tag{6.4}
\]

Since the first eigenvalue is the largest, it is easy to see that \( ||\Gamma - yy^T|| \) reaches the minimum when \( y \) is the first covariance vector. This completes the proof.

Thus, we state that the first covariance vector is one of the best choices for the vector \( \delta_m \). We then simplify the data summary as follows.

**Definition 6.3** For the data summary \( DS_m = \{n_m, \nu_m, \Gamma_m\} \) for the \( m^{th} \) subcluster, its **simplified data summary** \( s_m \) is \( \{n_m, \nu_m, \delta_m\} \) where \( \delta_m = \sqrt{\lambda_1} c_1 \) is the first covariance vector of \( (\Gamma_m - \nu_m \nu_m^T) \).

The simplified data summary contains only a number and two vectors. Based on the simplified data summary representation and the observation in Eq.(6.2), we then derive EMADS.

### 6.1.2 The EMADS Clustering Algorithm

We aim to generate a Gaussian mixture model \( \Phi \) from the simplified data summaries directly. Since we do not know the value of each data item, we can not use the Gaussian density function. We need to define a new density function \( \psi \) only based on which subcluster a data item \( x_i \) belongs to. With this density function, we then may have a new mixture model \( \Psi \) for subclusters. The new mixture \( \Psi \)
has the same parameters with the Gaussian mixture model $\Phi$. So, the problem of finding good parameters for the Gaussian mixture model $\Phi$ becomes searching for those for the mixture model $\Psi$.

Considering the first observation above, we give the new density function.

**Definition 6.4** A single data item $x_i$ within the $m^{th}$ subcluster, its probability under the probability density function $\psi$ is

$$
\psi(x_i | m) = \frac{\exp \left\{-\frac{1}{2} [s_m^T \Sigma_k^{-1} s_m + (\nu_m - \mu_k)^T \Sigma_k^{-1} (\nu_m - \mu_k)] \right\}}{(2\pi)^{D/2} |\Sigma_k|^{1/2}},
$$

(6.5)

where $\theta_k = (\mu_k, \Sigma_k)$ is the parameters for the $k^{th}$ component of $\Psi$.

It is worth pointing out that $\theta_k = (\mu_k, \Sigma_k)$ are also the parameters for the $k^{th}$ Gaussian component. If $\delta_m = 0$, the density function in Eq.(6.5) is equivalent to a Gaussian distribution. In general, however, it is not a genuine density function since its integral over the whole data space is less than 1. Roughly speaking, if the subcluster variance, $n_m (\Gamma_m - \nu_m \nu_m^T)$, is small, the item $\delta_m^T \Sigma_k^{-1} \delta_m$ is small too. In other words, this density function has a high probability for a data item in dense areas, which accords to Gaussian distributions. So, the modification is reasonable.

With this, we get a density function under the new mixture model $\Psi$ for $x_i$ within the $m^{th}$ subcluster:

$$
p(x_i | \Psi) = p(s_m | \Psi) = \sum_{k=1}^{K} p_k \psi(s_m | \mu_k, \Sigma_k).
$$

(6.6)

The log-likelihood, which measures the fitness of the mixture model over the subclusters, may be defined as

$$
L(\Psi) = \sum_{i=1}^{N} \log p(x_i | \Psi) = \sum_{m=1}^{M} n_m p(s_m | \Psi)
$$

(6.7)

$$
= \sum_{m=1}^{M} n_m \cdot \log \left( \sum_{k=1}^{K} p_k \psi(s_m | \mu_k, \Sigma_k) \right)
$$
Since the mixture model $\Psi$ has the same parameters with the Gaussian mixture model $\Phi$, we may find a good solution of $\Phi$ by finding a good solution of $\Psi$. We now derive a new EM algorithm to efficiently get a mixture model with maximum likelihood. If a data item $x_i$ is in the $k^{th}$ cluster, its indicator vector $z_i = [z_{i1}, z_{i2}, \cdots, z_{iK}]^T$ is equal to zero except $z_{ki} = 1$. Then the complete data vector $y_i = (x_i^T, z_i^T)^T$, which is augmented by a cluster indicator. The likelihood of the fully labelled data, $y_i$, is

$$g(y_i|\Psi) = p(x_i|z_i, \Psi)p(z_i|\Psi) = \psi(x_i|\theta_k)p_k = \prod_{k=1}^{K}[\psi(x_i|\theta_k)p_k]^{z_{ki}}.$$ 

The last equation holds since $z_{ki}$ is either zero or one. For the $N$ data items, we have

$$g(y_1, \cdots, y_N|\Psi) = \prod_{i=1}^{N}\prod_{k=1}^{K}[\psi(x_i|\theta_k)p_k]^{z_{ki}} = \prod_{m=1}^{M}\prod_{k=1}^{K}[\psi(s_m|\theta_k)p_k]^{z_{kmn_m}} \quad (6.8)$$

The last equation is valid because that data items within the same subcluster have the same probability, and thus have the same cluster label. In other words, $\psi(x_i|\theta_k) = \psi(x_j|\theta_k)$ and $z_i = z_j$ if $x_i$ and $x_j$ fall into the same subcluster.

The log-likelihood $L(\Psi)$ is obtained from $g(\{y\}|\Psi)$ by integrating over all possible $\{y\}$ in which the set $\{x\}$ is embedded:

$$L(\Psi) = \log [p(\{x\}|\Psi)] = \int \log [g(\{y\}|\Psi)] dz = \sum_{m=1}^{M} n_m \log p(s_m|\Psi).$$

This agrees with the log-likelihood definition in Eq.(6.7).

Now let us calculate the expectation of the complete data log-likelihood, conditional on the observed data $\{x\}$ (which is replaced by $\{s\}$ literally) and the current value of the parameters, $\Psi^{(j)}$.

$$Q(\Psi; \Psi^{(j)}) = E\left[\log(g(\{y\}|\Psi))|\{x\}, \Psi^{(j)}\right] = E\left[\log(g(\{y\}|\Psi))|\{s\}, \Psi^{(j)}\right]$$

$$= \sum_{m=1}^{M} n_m \sum_{k=1}^{K} E\left[z_{km}|\{s\}, \Psi^{(j)}\right] \left[\log p_k + \log(\psi(s_m|\mu_k^{(j)}, \sigma_k^{(j)}))\right]$$

$$\triangleq \sum_{i=1}^{M} n_m \sum_{k=1}^{K} r_{mk} \left[\log f_k^{(j)} + \log(\psi(s_m|\mu_k^{(j)}, \sigma_k^{(j)}))\right] \quad (6.9)$$
where
\[ r_{mk} = E[z_{km}|\{s\}, \Psi^{(j)}] = \frac{\sum_{l=1}^{K} p_l^{(j)} \psi(s_m|\mu_l^{(j)}, \Sigma_l^{(j)})}{\sum_{l=1}^{K} p_l^{(j)} \psi(s_m|\mu_l^{(j)}, \Sigma_l^{(j)})}. \quad (6.10) \]

Now we turn to maximize \( Q(\Psi; \Psi^{(j)}) \) with respect to \( \Psi \), and consider the parameters \( p_k, \mu_k, \) and \( \Sigma_k \) in turn. We need to introduce a Lagrange multiplier \( \lambda \) to handle the constraint \( \sum_{k=1}^{K} p_k = 1 \). Differentiating \( Q(\Psi; \Psi^{(j)}) - \lambda \left( \sum_{k=1}^{K} p_k - 1 \right) \) with respect to \( p_k \), we get
\[ \sum_{m=1}^{M} n_m r_{mk} \frac{1}{p_k} - \lambda = 0 \quad \text{for} \quad k = 1, \ldots, K. \]
Summing up these \( K \) equations together, we have
\[ \lambda \sum_{k=1}^{K} p_k = \sum_{k=1}^{K} \sum_{m=1}^{M} n_m r_{mk} = \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} r_{mk} \right] = N. \]
This leads to
\[ \lambda = N, \quad \text{(6.11)} \]
\[ \hat{p}_k = \frac{\sum_{m=1}^{M} n_m r_{mk}}{N}. \]

For the parameters of Gaussian components, we have partial derivatives on the density function:
\[ \frac{\partial \log \psi(s_m|\mu_k, \Sigma_k)}{\partial \mu_k} = (\Sigma_k)^{-1} (\nu_m - \mu_k), \quad \text{and} \]
\[ \frac{\partial \log \psi(s_m|\mu_k, \Sigma_k)}{\partial \Sigma_k} = -\frac{1}{2} \Sigma_k^{-1} + \frac{1}{2} \Sigma_k^{-1} (\delta_m \delta_m^T + (\nu_m - \mu_k)(\nu_m - \mu_k)^T) \Sigma_k^{-1}. \]
Here we employ\(^1\), for any symmetric matrix \( A \),
\[ \frac{\partial |A|^{-1/2}}{\partial A} = -\frac{1}{2} |A|^{-3/2} \cdot \frac{\partial |A|}{\partial A} = -\frac{1}{2} |A|^{-3/2} \cdot |A| (A^{-1})^T \quad (6.12) \]
\[ = -\frac{1}{2} |A|^{-1/2} A^{-1}, \quad (6.13) \]
\(^1\)These equations may be found in [74, p. 229 and p. 234].
and

\[
\frac{\partial a^T A^{-1} a}{\partial A} = -A^{-1} a a^T A^{-T} = -A^{-1} a a^T A^{-1}.
\]

(6.14)

Differentiating \( Q(\Psi; \Psi^{(j)}) \) with respect to \( \mu_k \) and equating the partial differential to zero gives

\[
\frac{\partial Q(\Psi; \Psi^{(j)})}{\partial \mu_k} = \sum_{m=1}^{M} n_m r_{mk} \frac{1}{2\sigma_{dk}} (\nu_m - \mu_k) = 0.
\]

This gives the re-estimation for \( \mu_k \) as

\[
\hat{\mu}_k = \frac{\sum_{m=1}^{M} n_m r_{mk} \nu_m}{\sum_{m=1}^{M} n_m r_{mk}}.
\]

(6.15)

Similarly, differentiating \( Q(\Psi; \Psi^{(j)}) \) with respect to \( \Sigma_k \) and equating the result to zero leads to

\[
\hat{\Sigma}_k = \frac{\sum_{m=1}^{M} n_m r_{mk} \left[ \delta_m \delta_m^T + (\nu_m - \mu_k)(\nu_m - \mu_k)^T \right]}{\sum_{m=1}^{M} n_m r_{mk}}
\]

(6.16)

According to Theorem 6.2, the above equation approximates the second observation in Eq.(6.2) in the extreme. So, our EM algorithm unifies the two observations. Thus, we may get a good Gaussian mixture model through the new EM algorithm. We rewrite the algorithm as follows.

**Algorithm 6.5 (EMADS)**

1. **Initialization:** Set the current iteration \( j \) to 0 and initialize the parameters in the mixture model: \( p^{(j)}_k (> 0), \mu^{(j)}_k \) and \( \Sigma^{(j)}_k \) such that \( \sum_{k=1}^{K} p^{(j)}_k = 1 \) and \( \Sigma^{(j)}_k \) is a positive definite matrix \( (k = 1, \cdots, K) \). Here the number of clusters \( K \) is given in advance.
2. **E-step**: Given the mixture model $\Psi^{(j)}$, compute the membership probability $r_{mk}^{(j)}$ for $s_m$:

$$r_{mk}^{(j)} = \frac{P_k^{(j)} \psi(s_m | u_k^{(j)}, \Sigma_k^{(j)})}{\sum_{i=1}^{K} P_i^{(j)} \psi(s_m | u_i^{(j)}, \Sigma_i^{(j)})}$$  \hspace{1cm} (6.17)

3. **M-step**: given $r_{mk}^{(j)}$, update the mixture model parameters for $k = 1, \ldots, K$:

$$p_k^{(j+1)} = \frac{1}{N} \sum_{m=1}^{M} n_m r_{mk}^{(j)}$$  \hspace{1cm} (6.18)

$$\mu_k^{(j+1)} = \frac{\sum_{m=1}^{M} n_m r_{mk}^{(j)} \nu_m}{\sum_{m=1}^{M} n_m r_{mk}^{(j)}} = \frac{\sum_{i=1}^{M} n_m r_{mk}^{(j)} \nu_m}{N \cdot p_k^{(j+1)}}$$  \hspace{1cm} (6.19)

$$\Sigma_k^{(j+1)} = \frac{\sum_{m=1}^{M} n_m r_{mk}^{(j)} \left[ \delta_m \delta_m^T + (\nu_m - \mu_k^{(j)}) (\nu_m - \mu_k^{(j)})^T \right]}{N \cdot p_k^{(j+1)}}$$  \hspace{1cm} (6.20)

4. **Termination**: if $|L(\Psi^{(j+1)}) - L(\Psi^{(j)})| \geq \epsilon |L(\Psi^{(j)})|$, set $j$ to $j + 1$ and go to step 2.

EMADS is a little different from the classical EM algorithm for Gaussian mixture models. It is also easy to implement.

### 6.1.3 Convergence Guarantee

EMADS is terminated when the log-likelihood values $L(\Psi^{(j)})$ changes slowly. This criterion must be satisfied as supported by the following convergence theorem.

**Theorem 6.6** If the matrix $[\nu_1, \nu_2, \ldots, \nu_M]$ is full rank, the log-likelihood $L(\Psi)$ for EMADS converges monotonically to some log-likelihood value $L^* = L(\Psi^*)$ for some stationary mixture model $\Psi^*$.

**Proof**: First of all, we prove the feasibility of EMADS. By induction, we show that the covariance matrix $\Sigma_k^{(j)}$ in Eq.(6.20) is always positive definite, and $p_k^{(j)}$ and $r_{mk}^{(j)}$ are positive during the execution.
It is right in the initialization, $\Sigma_k^{(j)}$ is initialized to be positive definite. Once $\Sigma_k^{(j)}$ is positive definite, $r_{mk}^{(j)}$ is larger than zero according to Eq.(6.17).

If $\Sigma_k^{(j)}$ is positive definite, and $p_k^{(j)}$ and $r_{mk}^{(j)}$ are positive in iteration $j$, then we prove they are positive in the next iteration. Both $\delta_m^T \delta_m$ and $(\nu_m - \mu_k^{(j)})^T (\nu_m - \mu_k^{(j)})^T$ are positive semi-definite matrices. Considering that $[\nu_1, \nu_2, \cdots, \nu_M]$ is full rank, the matrix $\Sigma_k^{(j+1)}$ must be positive definite. Because of this, it is clear to see that $r_{mk}^{(j+1)}$ is always positive. So does $p_k^{(j+1)}$.

After proving the feasibility of the algorithm, we now prove the log-likelihood value converges to certain value. In the derivation procedure in Section 6.1.2, we follow the general EM algorithm to get EMADS, and thus EMADS is an instance of the general EM algorithm. The monotonous property of the log-likelihood of the general EM algorithm is applicable, i.e., $L(\Psi^{(j+1)}) \geq L(\Psi^{(j)})$. Furthermore, since $\delta_m^T \Sigma_k^{-1} \delta_m \geq 0$, $\psi(s_m|\theta_k) \leq \phi(\nu_m|\theta_k)$ is bounded. Then

$$L(\Psi) \leq \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} p_k \psi(s_m|\theta_k) \right] \leq \sum_{m=1}^{M} n_m \left[ \sum_{k=1}^{K} p_k \phi(\nu_m|\theta_k) \right]$$

has an upper bound too. Thus, $L(\Psi^{(j)})$ converges monotonically to some value $L(\Psi^*)$.

Finally we show that the limit of $\{\Psi^{(j)}\}$ is a stationary mixture model. The function $Q(\Phi; \Psi)$ in Eq.(6.9) consists of only arithmetic and logarithm operations. So, it is clear that the function $Q(\Phi; \Psi)$ is continuous in both $\Phi$ and $\Psi$. Recall that the convergence theorem about the general EM algorithm, i.e., theorem 3.2 in [64, p.88], which says that all the limit records of any instance of the general EM algorithm are stationary records of $L(\Psi)$ if the function $Q(\Phi; \Psi)$ is continuous in both $\Phi$ and $\Psi$. So, the limit mixture model of $\{\Psi^{(j)}\}$ is a stationary mixture model. The proof of the theorem is completed.

\subsection*{6.1.4 Complexity}

Let us discuss the complexity of EMADS. In the E-step, it needs to calculate $M \times K$ membership probabilities $r_{mk}^{(j)}$. For the calculation of each $r_{mk}^{(j)}$, EMADS
has to calculate the probability under each component distribution according to Eq.(6.5). It involves $O(D^2)$ arithmetic operations. Thus the E-step of EMADS takes $O(MKD^2)$ operations. Similarly, the M-step of EMADS takes $O(MKD^2)$. In a word, the computational complexity of EMADS is $O(MKD^2I)$ where $I$ is the number of iterations. So, it is linear with the number of data summaries.

EMADS requires to store $M(2D + 1)$ floating point numbers for the simplified data summaries, $MK$ floating point numbers for the membership, and $K(D^2 + D + 1)$ floating point numbers for the mixture model. Thus, the total storage requirement of EMADS is $2MD + MK + KD^2 + KD + K + M$ floating point numbers. Consequently, the storage requirement of EMADS is independent to the number of data items, $N$. In other words, given a data set, we can choose an appropriate number of subclusters, $M$, to summarize the data set into the given main memory.

6.2 Performance of EMADS

6.2.1 Methodology

Working on the data summaries generated by the adaptive grid-based data summarization procedure, EMADS can be used to establish a scalable clustering system. We call it the gEMADS algorithm hereafter. The gEMADS algorithm is mainly used to examine the sensitivity of the EMADS algorithm to different data summary granularity since we can manipulate the grid-based data summarization procedure easily to get some very skew cases. Similarly, combining EMADS with the BIRCH’s data summarization procedure, a new scalable model-based clustering system may be established. We call it the bEMADS algorithm.

To illustrate the performance of the gEMADS and bEMADS algorithms, we compare them with several sampling or data summarization based clustering algorithms. They include
The EM algorithm: It is the classical EM algorithm for Gaussian mixture models.

The sampling EM algorithm: It is the EM algorithm working on 5% random samples. It is referred to as sampEM hereafter.

The gEMACF and the bEMACF algorithms: These algorithms are discussed in Chapter 5. They assume that the data attributes are statistically independent.

We use two termination criteria for all algorithms. One is the maximal number of iterations, 500. The clustering algorithms are also terminated if the successive log-likelihood modification is within $10^{-5}$ of current value as used in [65, 83]. In other words, $\epsilon$ is $10^{-5}$ in EMADS, EMACF and the EM algorithms. All these algorithms are coded in MATLAB and experiments are conducted on a Sun Enterprise E4500 server. These model-based clustering algorithms are initialized with the cluster centers generated by K-means from the weighted samples. All experimental results reported are averaged on 10 independent runs except that the results for the EM algorithm are based on three runs since it takes too long to run. When the BIRCH’s data summarization procedure is used to generate subclusters, the default parameter settings of BIRCH are used except for the following three parameters. The maximum main memory used is 8,192,000 bytes, the outliers are defined as the leaf entries containing less than 10% of the average number of data items per leaf entry, and the input range is 4,000. In other words, the maximum number of subclusters is 4,000.

6.2.2 Evaluation Metrics and Data Generation

Since all algorithms finally generate Gaussian mixture models, the natural evaluation metric for their log-likelihood value. We take 10% random samples to examine the generated Gaussian mixture models. To be more observable, we
average the log-likelihood over the samples.

We also use the clustering accuracy to measure the generated mixture models for the synthetic data sets. In this case, the clustering results generated by the original mixture model are used as references. The clustering accuracy is defined as the proportion of samples which are correctly clustered by the generated mixture model [65].

We generate two groups of synthetic data sets based on random mixture models. The first group has three data sets on a 2-dimensional space. For each mixture model, the means of the Gaussian components are located on grids and are from \([1,1], [2,1], \ldots, [\sqrt{K}, \sqrt{K}]\). The mixing proportion \(p_k\) varies in \([\frac{1}{100K}, \frac{3}{K}]\), and the cluster sizes may be very skew. The variance for each attribute fall in \([0.001, 0.5]\). The correlation values are randomly generated to ensure that each covariance matrix is positive definite. Some samples of these three data sets are illustrated in Figures 6.4(a), 6.4(b) and 6.4(c). The other group has four data sets which are generated according to random Gaussian mixture models. The main difference from the first group is their mean vector generation. A pair of mean vectors are generated together to ensure that their Euclidean distance is 1.0. In other words, this pair of clusters are very close and are not well separated. A typical data set is illustrated in Figure 6.4(d). In these seven data sets, the number of data items ranges from 108,000 to 1,100,000, and the number of clusters ranges from 9 to 20. All these parameters can be found in Table 6.1, where \(N, D, K\) and \(M\) indicate the number of data items, the data dimensionality, the number of clusters, and the number of subclusters, respectively.

### 6.2.3 Sensitivity

We first discuss the influence of the data summarization procedure on EMADS. The first data set shown in Figure 6.4(a) is taken as an example. Figure 6.5 illustrates the clustering accuracy of three clustering algorithms for different data
Figure 6.4: Data samples of the first four data sets. (a) A 10% sample of the first data set, (b) A 5% sample of the second data set, (c) A 5% sample of the third data set, and (d) A 10% sample of the fourth data set.
Figure 6.5: The sensitivity of clustering algorithms to the data summarization or sampling procedure.

summarization or sampling procedures. The data summarization procedures are manipulated by different grid structures. For the first grid structure, we partition two attributes into 56 segments respectively and get 3,136 cells. The grid structure is indicated by 56*56 in Figure 6.5. The sampEM(M) algorithm refers to the classical EM algorithm working on M samples, where M is equal to the number of subclusters.

For the first seven grid structures, the segment numbers for each attribute are 56, 48, 40, 32, 24, 16 and 12, respectively. In other words, the cell granularity increases gradually. As shown in Figure 6.5, the clustering accuracy of all three algorithms decreases roughly. Because the two attributes are correlated, the independent Gaussian mixture model is not suitable for this data set. The clustering accuracy of the gEMACF algorithm is not so good. It performs much worse than the other two algorithms. The clustering accuracy values of the gEMADS algorithm range from 95.4% to 91.4% for the first five grid structures. The clustering
Figure 6.6: Two typical mixture models generated by the gEMADS algorithm from different data summarization procedures. An ‘o’ and its associated ellipse represent a generated Gaussian distribution component. A ‘+’ and its associated dotted ellipse indicates an original Gaussian distribution component. A ‘*’ represents a data summary.
results of the gEMADS algorithm are acceptable when the segment number is larger than 24. The clustering accuracy value of the gEMADS algorithm for the 16*16 grid structure is 81.6%. It is not so good but better than the two counterparts. A typical mixture model generated by the gEMADS algorithm for the grid structure can be found in Figure 6.6(a). In the figure, an ‘o’ and its associated ellipse represent a generated Gaussian distribution component. A ‘+’ and its associated dotted ellipse indicates an original Gaussian component. A ‘*’ represents a data summary. The generated mixture model is close to the original one.

The last three grid structures aim to generate very skew subclusters. For example, the 12*86 grid structure divides the two attributes into 12 and 86 segments respectively. So the cell width is 7.2 times of the cell height as the small rectangles shown in Figure 6.6(b). For this grid structure, the clustering accuracy value of the gEMACF algorithm is 77.2%. The clustering accuracy of the sampEM(M) algorithm is 76.0% where about 1,032 samples are taken. As a comparison, the clustering accuracy of the gEMADS algorithm is as high as 83.6%. A typical mixture model generated by the gEMADS algorithm may be found in Figure 6.6. Although the gEMADS algorithm degenerates with clustering accuracy values from 92.0% to 83.6% for the last three grid structures, the degeneration is not very serious.

In summary, the gEMADS algorithm is not very sensitive to the data summarization procedure. Its performance is good if the subcluster granularity is not too large and subclusters are not too skew.

### 6.2.4 Clustering Accuracy

Table 6.1 lists clustering accuracy of four algorithms for the seven synthetic data sets. In the table, the bold text denotes the best one for the corresponding data set. \( N, D, K \) and \( M \) indicate the number of data items, the data dimensionality,
Table 6.1: The clustering accuracy of the four clustering algorithms on the seven synthetic data sets. \(N\), \(D\), \(K\) and \(M\) indicate the number of data items, the data dimensionality, the number of clusters, and the number of subclusters, respectively.

<table>
<thead>
<tr>
<th>DataSet</th>
<th>(N)</th>
<th>(D)</th>
<th>(K)</th>
<th>(M)</th>
<th>bEMADS</th>
<th>bEMACF</th>
<th>EM</th>
<th>sampEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>108000</td>
<td>2</td>
<td>9</td>
<td>2986</td>
<td>0.941</td>
<td>0.853</td>
<td>0.987</td>
<td>0.909</td>
</tr>
<tr>
<td>2</td>
<td>500000</td>
<td>2</td>
<td>12</td>
<td>2499</td>
<td>0.908</td>
<td>0.782</td>
<td>0.862</td>
<td>0.839</td>
</tr>
<tr>
<td>3</td>
<td>1100000</td>
<td>2</td>
<td>20</td>
<td>3818</td>
<td>0.912</td>
<td>0.729</td>
<td>0.885</td>
<td>0.846</td>
</tr>
<tr>
<td>4</td>
<td>200000</td>
<td>2</td>
<td>10</td>
<td>2279</td>
<td>0.869</td>
<td>0.671</td>
<td>0.890</td>
<td>0.823</td>
</tr>
<tr>
<td>5</td>
<td>200000</td>
<td>3</td>
<td>10</td>
<td>3227</td>
<td>0.916</td>
<td>0.720</td>
<td>0.950</td>
<td>0.903</td>
</tr>
<tr>
<td>6</td>
<td>240000</td>
<td>4</td>
<td>12</td>
<td>3982</td>
<td>0.843</td>
<td>0.767</td>
<td>0.876</td>
<td>0.890</td>
</tr>
<tr>
<td>7</td>
<td>280000</td>
<td>5</td>
<td>14</td>
<td>2391</td>
<td>0.871</td>
<td>0.810</td>
<td>0.890</td>
<td>0.868</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.894</td>
<td>0.762</td>
<td>0.906</td>
<td>0.868</td>
</tr>
</tbody>
</table>

the number of clusters, and the number of subclusters, respectively. The data size ranges from 108,000 to 1,100,000. These parameters can be found in the first 5 columns of Table 6.1.

First, we examine the performance of the four algorithms on the largest data set with 1,100,000 data items (data set 3). A 5% sample of the data set is illustrated in Figure 6.4(c). The BIRCH’s data summarization procedure generates 3,818 subclusters. The clustering accuracy value of the bEMADS algorithm is 91.2%. They are 88.5% and 84.6%, respectively, for the EM and sampEM algorithms. The clustering accuracy value of the bEMADS algorithm is about 2.7% higher than that of the EM algorithm, and 6.9% higher than that of sampEM. The clustering accuracy of the bEMACF algorithm is 72.9%, which lags far behind the three other algorithms.

For the seven data sets, the bEMADS algorithm generates most accurate results on two cases. The average clustering accuracy value of the bEMADS algorithm is 89.4%. The average clustering accuracy values of the bEMACF, the EM and the sampEM algorithms are, respectively, 76.2%, 90.6%, and 86.8%. So, the bEMADS algorithm performs better than the bEMACF and the sampEM algorithms. It can generate clustering results very close to those of the EM algorithm.
6.2.5 Scalability

Figure 6.7 illustrates the execution times for the four clustering algorithms on eight 4-dimensional data sets. All data sets have 10 clusters. These data sets differ only in their sizes. Their data sizes increase exponentially from 6,250 to 800,000. To clearly show the scalability, logarithm axes are used in the figure. The numbers of subclusters vary from 1,577 to 3,044. The parameters for these eight data sets as well as the clustering accuracy of the four algorithms can be found in Table 6.2.

It can be observed from Figure 6.7 that the execution times of the bEMADS algorithm increase very slowly. It takes 673.8 seconds for the data set with 6,250 data items, and takes 1,106.2 seconds for the data set with 800,000 data items. The execution time of the bEMADS algorithm is little dependent on the number of data items since the data summarization procedure executes efficiently. The whole execution time of the bEMADS algorithm mainly spends on the mixture model generation. For example, for the largest data set, the data summarization procedure takes 255.2 seconds and the EMADS algorithm takes 851.0 seconds.

The bEMACF algorithm is also scalable. Its execution times range from 176.6 seconds for the smallest data set to 509.6 seconds for the largest data set. Because the bEMACF algorithm assumes the data attributes are statistically independent, the bEMACF algorithm has a lower computational complexity than the bEMADS algorithm. On the other hand, as listed in Table 6.2, the bEMADS algorithm generates clustering results with higher accuracy than those generated by the bEMACF algorithm for all eight data sets. The average clustering accuracy of the bEMADS algorithm is 93.5%, which 2.0% higher than the value of 91.5% of the bEMACF algorithm.

The execution times of sampEM range from 125.2 seconds to 12,071.0 seconds as shown in Figure 6.7. The execution times increase almost linearly with the data size, since sampEM has no restriction on the amount of the main memory.
Figure 6.7: Execution times of the four clustering algorithms for eight 4-dimensional data sets

used during the execution. For the first four small data sets, sampEM runs faster than the bEMADS algorithm. It is caused by two factors. The first one is the numbers of samples are smaller than the numbers of data summaries. The second one is the data summarization overhead. For the larger data sets, the data summarization overhead becomes relatively smaller, and the bEMADS algorithm runs much faster than sampEM. The ratios of the execution times of sampEM to those of the bEMADS algorithm are 5.6 and 10.9, respectively, for the data sets with 400,000 and 800,000 data items. In addition, the clustering accuracy values of the bEMADS algorithm are normally better than those of sampEM. As listed in Table 6.2, the average clustering accuracy of the bEMADS algorithm is 5.5% higher than the value of 88.0% for sampEM.

The execution times of the classical EM algorithm increase from 2,344.4 seconds for the data set with 6,250 data items to 359,966.7 seconds for the largest
Table 6.2: The clustering accuracy of the four clustering algorithm on eight 4-dimensional data sets.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>(N (\text{1000}))</th>
<th>(D)</th>
<th>(K)</th>
<th>(M)</th>
<th>bEMADS</th>
<th>bEMACF</th>
<th>EM</th>
<th>sampEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSa</td>
<td>6.25</td>
<td>4</td>
<td>10</td>
<td>1577</td>
<td>0.920</td>
<td>0.919</td>
<td>0.929</td>
<td>0.772</td>
</tr>
<tr>
<td>DSb</td>
<td>12.5</td>
<td>4</td>
<td>10</td>
<td>2495</td>
<td>0.951</td>
<td>0.942</td>
<td>0.938</td>
<td>0.803</td>
</tr>
<tr>
<td>DSc</td>
<td>25</td>
<td>4</td>
<td>10</td>
<td>2721</td>
<td>0.942</td>
<td>0.897</td>
<td>0.939</td>
<td>0.926</td>
</tr>
<tr>
<td>DSD</td>
<td>50</td>
<td>4</td>
<td>10</td>
<td>3074</td>
<td>0.923</td>
<td>0.909</td>
<td>0.846</td>
<td>0.837</td>
</tr>
<tr>
<td>DSe</td>
<td>100</td>
<td>4</td>
<td>10</td>
<td>2951</td>
<td>0.967</td>
<td>0.935</td>
<td>0.910</td>
<td>0.939</td>
</tr>
<tr>
<td>DSl</td>
<td>200</td>
<td>4</td>
<td>10</td>
<td>2951</td>
<td>0.970</td>
<td>0.946</td>
<td>0.956</td>
<td>0.921</td>
</tr>
<tr>
<td>DSG</td>
<td>400</td>
<td>4</td>
<td>10</td>
<td>2582</td>
<td>0.889</td>
<td>0.863</td>
<td>0.970</td>
<td>0.969</td>
</tr>
<tr>
<td>DSh</td>
<td>800</td>
<td>4</td>
<td>10</td>
<td>2763</td>
<td>0.916</td>
<td>0.908</td>
<td>0.931</td>
<td>0.875</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.935</td>
<td>0.915</td>
<td>0.927</td>
<td>0.880</td>
</tr>
</tbody>
</table>

data set with 800,000 data items. The execution times of the EM algorithm increase almost linearly with the data size. It is because that the amount of the main memory used is not restricted during the execution. For the data set with 800,000 data items, bEMADS runs 339.0 times faster than the EM algorithm. For the first data set with 6,250 data items, the speedup factor is about 3.5. For the other data sets, the speedup factors of the bEMADS algorithm to the EM algorithm range from 6.1 to 202.3. Thus, the bEMADS algorithm can run one or two orders of magnitude faster than the EM algorithm. In addition, it can be observed from the clustering accuracy values listed in Table 6.2 that EMADS also can generate more accurate clustering results than the EM algorithm on several data sets. The average clustering accuracy of the bEMADS algorithm is 93.5%. It is a little higher than the value of 92.7% of the EM algorithm.

### 6.2.6 Application on Real World Data Sets

For the real world data sets, we use the log-likelihood to measure the accuracy of the generated Gaussian mixture models. The larger the log-likelihood is, the better the match between the mixture model and a data set is. We still use the three real work data sets used in Chapter 5.

The first data set, the Forest CoverType Data, is downloaded from the UCI Knowledge Discovery in Databases Archive (http://kdd.ics.uci.edu). It describes
Table 6.3: The performance of the bEMADS algorithm on the Forest CoverType Data.

<table>
<thead>
<tr>
<th></th>
<th>bEMACF</th>
<th>bEMADS</th>
<th>sampEM(15%)</th>
<th>sampEM(5%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>-3.242 ± 0.019</td>
<td>-3.083 ± 0.011</td>
<td>-3.078 ± 0.017</td>
<td>-3.086 ± 0.018</td>
</tr>
<tr>
<td>Time(Sec.)</td>
<td>2064.9 ± 891.2</td>
<td>7985.5 ± 3635.2</td>
<td>173672.5 ± 80054.2</td>
<td>49745.8 ± 10328.9</td>
</tr>
</tbody>
</table>

forest cover for 30x30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The data set has 581,012 data items. We use five quantitative attributes including Elevation, Aspect, Horizontal Distance To Hydrology, Horizontal Distance To Roadways, and Horizontal Distance To Fire Points. All attribute values are scaled into [0, 3]. We use a Gaussian mixture model with 15 components to describe the data set. The BIRCH’s data summarization procedure generates 3,186 subclusters.

Table 6.3 lists the experimental results which include the standard deviations of the log-likelihood and execution time. Here sampEM(15%) represents the classical EM algorithm working on 15% random samples and sampEM(5%) represents the classical EM algorithm working on 5% random samples. We used sampEM(15%) for comparison since the EM algorithm cannot generate a mixture model after running for 100 hours. In fact, even sampEM(15%) takes 173,672.5 seconds (about 48.2 hours), and sampEM(5%) needs 49,745.8 seconds for execution.

On average, the bEMADS algorithm takes about 7,985.5 seconds. It runs 21.7 times faster than sampEM(15%), and 6.2 times faster than sampEM(5%). The log-likelihood value of the bEMADS algorithm is -3.083. The log-likelihood values of sampEM(15%) and sampEM(5%) are -3.078 and -3.086, respectively. The log-likelihood of the bEMADS algorithm is lower than that of sampEM(15%) by 0.005 and is higher than that of sampEM(5%) by 0.003. Their differences are not so large in comparison with the log-likelihood standard deviation, which is about 0.011.
Table 6.4: The performance of the bEMADS algorithm on the Census-Income Database.

<table>
<thead>
<tr>
<th></th>
<th>Census</th>
<th>bEMACF</th>
<th>bEMADS</th>
<th>EM</th>
<th>sampEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>-0.747 ± 0.003</td>
<td>-0.741 ± 0.004</td>
<td>-0.740 ± 0.004</td>
<td>-0.743 ± 0.006</td>
<td></td>
</tr>
<tr>
<td>Time (Sec.)</td>
<td>1361.3 ± 371.6</td>
<td>4283.9 ± 1040.1</td>
<td>495056.4 ± 87312.1</td>
<td>16359.4 ± 5873.3</td>
<td></td>
</tr>
</tbody>
</table>

Because data attributes are assumed to be independent in EMACF, EMACF has a lower computational complexity than EMADS. The bEMACF algorithm runs fastest among the four algorithms. However, it generates the worst mixture models with the log-likelihood value of -3.242. It is lower than that of the bEMADS algorithm by 0.159. The difference is larger than the log-likelihood standard deviation.

The second data set, the Census-Income Database, is also downloaded from the UCI KDD archive. This data set contains weighted census data extracted from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. The data set contains 299,285 data items. We used 3 continuous attributes including Age, Dividends From Stocks, and Weeks Worked In Year. All attribute values are scaled into [0, 3]. The mixture model used has 10 Gaussian distribution components. The BIRCH's data summarization procedure generates 3,836 subclusters.

Table 6.4 lists the performance of the four algorithms for the Census-Income Database. The classical EM algorithm needs 495,056.4 seconds for execution. The bEMADS algorithm takes 4,283.9 seconds. The ratio of the execution times is 115.6. The data set takes sampEM 16,359.4 seconds, which is 3.8 times longer than that of the bEMADS algorithm. On the other hand, The log-likelihood value of the bEMADS algorithm is -0.741. The log-likelihood values of the EM and the sampEM algorithms are -0.740 and -0.743, respectively. The log-likelihood of the bEMADS algorithm is lower than that of the EM algorithm by 0.001 and higher than that of sampEM by 0.002. The differences are not very significant in
Table 6.5: The performance of bEMADS on the California housing data.

<table>
<thead>
<tr>
<th>Housing</th>
<th>bEMACF</th>
<th>bEMADS</th>
<th>EM</th>
<th>sampEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-likelihood</td>
<td>3.512 ± 0.141</td>
<td>7.214 ± 0.191</td>
<td><strong>7.682 ± 0.149</strong></td>
<td>6.776 ± 0.239</td>
</tr>
<tr>
<td>Time(Sec.)</td>
<td>684.9 ± 113.2</td>
<td>3232.4 ± 525.8</td>
<td>16405.5 ± 2906.2</td>
<td>1433.9 ± 514.7</td>
</tr>
</tbody>
</table>

comparison with the log-likelihood standard deviation, which is about 0.004.

The bEMACF algorithm runs fastest among the four algorithms, even 3.1 times faster than the bEMADS algorithm. However, it generates the worst mixture models with the average log-likelihood value of -0.747, which is 0.006 lower than that of the bEMADS algorithm. The difference is larger than the log-likelihood standard deviation 0.004 of the bEMADS algorithm. This point is mainly due to the attribute independence assumption of the bEMACF algorithm.

The third data set, the California housing data, describes all the block groups in California from the 1990 USA Census. It is downloaded from www.spatial-statistics.com. It has 20,640 data items. In our experiments, we use 8 numeric attributes including Median House Value, Median Income, Total Rooms, Total Bedrooms, Population, Households, Latitude, and Longitude. All of the 8 attributes are scaled into [0, 3]. We use a mixture model of seven Gaussian components to describe the data set. The data summarization procedure of BIRCH generates 2,907 subclusters.

Table 6.5 lists the performance of the four clustering algorithms for the data set. The bEMADS algorithms takes about 3,232.4 seconds to generate mixture models with the average log-likelihood of 7.214. Its log-likelihood value is significantly larger than 3.512, the log-likelihood value of the bEMACF algorithm. The log-likelihood value of sampEM is 6.776. The difference is 0.458, which is almost twice of the log-likelihood standard deviation of sampEM. The bEMACF algorithm performs much better than sampEM. On the other hand, sampEM takes about 1,433.9 seconds. It runs faster than the bEMADS algorithm. The log-likelihood of the classical EM algorithm is 7.682, which is better than that
of the bEMADS algorithm. However, the EM algorithm takes 16,405.5 seconds. The bEMADS algorithm runs 5.1 times faster than the EM algorithm. For this small data set, the advantage of the bEMADS algorithm is not so apparent.

In summary, the bEMADS algorithm can run one or two orders of magnitude faster than the classical EM algorithm with little loss of accuracy. The bEMADS algorithm usually can generate more accurate mixture models than the 5% sampling EM algorithm with shorter execution time. The bEMADS algorithm can generate much more accurate mixture models than the bEMACF algorithm though the former needs a little longer execution time.

6.3 Discussion on In-memory Model-based Clustering Techniques

We have mentioned EMADS (the EM Algorithm for Data Summaries), EMACF (the EM Algorithm for Clustering Features), EMAWS (the EM Algorithm for Weighted Samples), ExEM (the Extended EM algorithm), and the classical EM algorithm for Gaussian mixture models in the thesis. All of them can generate Gaussian mixture models. However, their suitability for different kinds of summary statistics varies.

The EMADS algorithm works on simplified data summaries, which take into account the correlations among data attributes. This algorithm, deriving from the general EM algorithm, has a sound mathematical foundation.

Both EMACF and ExEM work on clustering features. The main difference is that EMACF is derived from the general EM algorithm, and thus also has a sound mathematical foundation. ExEM is established heuristically, and is out of the scope of the general EM algorithm [10].

EMAWS can assign different weights to the different samples. This is an advancement of the classical EM algorithm. If the weights are identical, EMAWS
Table 6.6: The computation complexity and the memory requirements of the in-memory clustering algorithms. \( N, D, K, I, \) and \( M \) indicate the total number of data items, the data dimensionality, the number of clusters, the number of iterations, and the number of subclusters, respectively.

<table>
<thead>
<tr>
<th></th>
<th>iEM</th>
<th>EMAWS</th>
<th>EMACF/ExEM</th>
<th>EM</th>
<th>EMADS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time complexity</td>
<td>( O(NDKI) )</td>
<td>( O(MKI) )</td>
<td>( O(MKI) )</td>
<td>( O(NKDDI) )</td>
<td>( O(MKDDI) )</td>
</tr>
<tr>
<td>Memory requirement</td>
<td>( N(D+K)+(D+K^2) )</td>
<td>( M(D+K+1)+(D+K^2) )</td>
<td>( M(2D+K+1)+(D+K^2) )</td>
<td>( N(D+K)+(D+K^2)+M(D+K) )</td>
<td>( M(D+K)+(D+K^2) )</td>
</tr>
</tbody>
</table>

is actually the classical EM algorithm.

The weighted samples have more powerful description than the samples, and a clustering feature can describe a subcluster better than a weighted sample. Furthermore, if data attributes are inherently independent, a simplified data summary becomes a clustering feature. So, it can be stated that EMADS is more general than EMACF and EMACF is more sophisticated than EMAWS.

EMACF and EMADS have new E-steps and M-steps. They are totally different from the classical EM algorithm for Gaussian mixture models. However, many EM variations are different from the classical EM algorithm only on either E-step or M-step. For example, the CEM [17, 18] and the re-weighted EM algorithms have the same M-step with the EM algorithm [90]. The lazy EM [83] and ExEM algorithms [10] have the same E-step with the EM algorithm.

We sum up the computation complexity and the memory requirements of these clustering algorithms in Table 6.6. EMAWS, EMACF, and ExEM have the same computation complexity. They are linear with the number of clustering features, the number of clusters, the data dimensionality, and the number of iterations. The computation complexity of two data summarization procedures is \( O(ND) \), so the total computation complexity of our scalable clustering systems, such as gEMACF, bEMACF, and bExEM, is \( O(ND + MDKI) \). Since the number of clustering features \( M \) is much smaller than the total number of data items \( N \), this computation complexity should be lower than the computation complexity of the classic EM algorithm (iEM), \( O(NDKI) \). For memory requirements,
EMACF/ExEM needs to store about $M(2D + K + 1) + K(2D + 1)$ floating point numbers. EMAWS needs to store about $M(D + K + 1) + K(2D + 1)$ floating point numbers. Because the data summarization procedures are restricted to operate within the given main memory, which may stores about $2M(2D + 1)$ floating point numbers, our scalable clustering systems can work within the maximum of the above bounds, i.e., $M(4D + K + 2) + K(2D + 1)$ floating point numbers. The memory requirement is much smaller than the value of $N(D + K) + K(2D + 1)$ for iEM since $M$ is usually much smaller than $N$. Similar situation can be found in the comparison between EM and EMADS. The main difference observed from Table 6.6 is that EMADS/EM are quadratic, while EMACF/iEM/EMAWS/ExEM is linear with the data dimensionality $D$.

Being different from the above iterative algorithms, the incremental EM algorithm [68] refines its mixture model incrementally. In fact, this on-line learning mode is also applicable to our EMACF and EMADS.

### 6.4 Chapter Summary

In this chapter, we have designed a simple data summarization representation which can take account of the correlations among data attributes, and developed EMADS (the EM Algorithm for Data Summaries) to generate mixture models from data summaries directly. The proposed algorithm, EMADS, has been proved to converge to a local maximum. Based on EMADS, we have established two scalable clustering systems: the gEMADS and bEMADS algorithms. Comprehensive experiments have shown that the bEMADS algorithm can run one or two orders of magnitude faster than the classical EM algorithms. Furthermore, it can generate mixture models with no or little loss of accuracy. The bEMADS algorithm is usually faster than the sampling EM algorithm and generates more accurate clustering results than the latter. We also have discussed that EMADS is a generalization of EMACF and the classical EM algorithm.
Chapter 7

Model Selection

This chapter gives a realization of the optional controller module of our model-based clustering framework discussed in Chapter 3. Though different kinds of manipulation information can be provided to the data summarization and the in-memory clustering modules, we concentrate on the number of clusters of a data set in this chapter.

There are many heuristics to choose an appropriate number of clusters. Some of them are based on some heuristic measures or statistical indices for cluster validity [60, 81, 94]. By balancing the model accuracy and complexity, the Bayesian Information Criterion (BIC) has been proposed and widely applied. It is defined as

\[
BIC(\Phi) = -2 \cdot L(\Phi) + v(\Phi) \cdot \log N
\]  

(7.1)

where \(N\) is the number of data items, \(v(\Phi)\) is the number of free parameters in the mixture model \(\Phi\), and \(L(\Phi)\) is the log-likelihood as defined in Section 2.1.2 of Chapter 2. The lower the BIC value is, the better the mixture model is. Experimental results have shown its good performance in practice [8, 30].

The enumeration strategy is the common procedure used to select the optimal number of clusters [9, 30, 90]. Given the number of clusters, it first uses a local search algorithm, say, the classical Expectation-Maximization (EM) algorithm, to find a good mixture model for the data set. Then these mixture models
with different numbers of clusters compete with one another based on a certain criterion, say the BIC \cite{9, 20, 30, 90}. It is referred to as the \textit{enumeration model-based clustering algorithm} (EnumEM) hereafter. The problem of this strategy is that it spends too much time on finding mixture models with inappropriate numbers of clusters. As an alternative, like in AutoClass \cite{20}, some promising numbers of clusters can be derived from the previous clustering results before invoking the local search algorithm. Both of them suffer from the local search property of the classical EM algorithm.

By introducing a better global search mechanism into model-based clustering techniques, we can improve the effectiveness in determining the number of clusters. The global search mechanism, say, a Genetic Algorithm (GA), can be used to explore the natural clusters among data and determine the optimal number of clusters automatically. This distinguishes our genetic-guided model-based clustering approaches from most of the previous research work on GA-guided clustering algorithms \cite{31, 40}, which mainly focuses on searching for the best clusters given the number of clusters. We give our genetic-guided model-based clustering techniques in the next section. Their performance is examined in Section 7.2. We discuss how to scale up the proposed algorithms for large data sets in Section 7.3.

### 7.1 Genetic-guided Model-based Cluster Analysis

For the last decade there has been a growing interest in evolutionary algorithms that are based on the Darwin’s theory of evolution. One of the implementations is Genetic Algorithms (GAs) \cite{36}. GAs perform a stochastic evolution process towards global optimization through the use of crossover, mutation, and selection operators. The search space of the problem is represented as a collection of individuals, which are referred to as chromosomes. The quality of a chromosome
is measured by a fitness function. After initialization, each generation produces and selects new children using the genetic operators. The process terminates when the maximum number of generation is reached or two consecutive generations do not produce noticeable population fitness improvement. We outline a canonical GA as follows.

1. Initialize the parameters such as the population size $\text{PopSize}$, the crossover probability $p_c$, the mutation probability $p_m$ and the maximal number of generation $\text{Gen}_{\text{max}}$; Generate the first population of chromosomes and initialize the current generation number $\text{gen}$ to 0;

2. Calculate the fitness value of each chromosome in the population;

3. Create the new population by repeating the following steps until the new population is filled up;

   (a) Select two parent chromosomes from the current population according to their fitness values where chromosomes with better fitness values have higher chances of being selected;

   (b) Perform crossover on the parents to form two new offsprings (children) with probability $p_c$; The parents do not go through the crossover operator would be reproduced as two new offsprings;

   (c) Mutate the new offspring with probability $p_m$;

4. If the termination condition is not satisfied, take the new generation of population as the current population and go back to Step 2; otherwise, return the best solution in the current population.

GAs are very flexible and can be adapted easily to the task on hand. Moreover, since they perform a global search, they are robust in handling difficult problem with several local optima. GAs have been widely used in cluster analysis [31, 40].
These algorithms have to specify the number of clusters in advance, which greatly impacts their utility in real world problems.

By combining GAs with several model-based clustering techniques, we can establish some genetic-guided model-based clustering algorithms. The global search capability of GAs enables the algorithms to focus on the promising solutions and avoid the search for solutions with an inappropriate number of clusters as early as possible. Our algorithms are outlined in Figure 7.1. The main structure is similar to a canonical genetic algorithm. However, four different genetic operators are
developed to enhance the performance. Different combinations of these operators result in different clustering algorithms. For simplicity, we call the algorithm with all genetic operators GAXEM, and the one without the HAC crossover operator GAEM. These genetic operators are described in detail in Sections 7.1.2 and 7.1.3.

7.1.1 Representation and Fitness

In our algorithms, each mixture model is coded as a chromosome to represent a clustering solution: the first gene is the number of clusters $K$, followed by genes representing the parameters of the clusters. The parameters for each cluster include the mixing proportion $p_k$, the mean vector $\mu_k$ and the covariance matrix $\Sigma_k$. Several typical individuals may be found in Figure 7.2. So, the length of the chromosome depends on the number of clusters and is varying during the execution of GAXEM/GAEM.

The fitness of a chromosome is evaluated by invoking an $n$-iteration EM algorithm. Starting from the mixture model indicated by a chromosome, the algorithm runs the E-step and the M-step for $n$ iterations. The updated model then replaces the old one, and its log-likelihood is used to calculate the $BIC$ value according to Eq. (7.1). The algorithms use a selection operator for minimization. In other words, the chromosome has a higher chance of being selected if its $BIC$ value is lower. In our implementation, the number of iterations, $n$, increases linearly with the generation number to increase the accuracy of the EM algorithm.

7.1.2 Crossover Operators

There are two crossover operators as shown in Figure 7.1. The first one is the two-point crossover operator used in several GAs. It exchanges the parameter values for some clusters between the two random crossover points.
Chapter 7 Model Selection

Figure 7.2: Illustration of the HAC crossover operator. A Gaussian distribution component’s parameters consist of $p_k$, $\mu_k$ and $\Sigma_k$.

The other one, the HAC crossover operator, is derived from the model-based Hierarchical Agglomerative Clustering (HAC) algorithm as discussed in Section 2.4. It is illustrated in Figure 7.2. Two parent mixture models are firstly merged into an interim mixture model by appending one model behind the other. The number of clusters in the interim individual is the sum of those of two parents. The mixture proportions of this interim mixture model are adjusted such that their sum is 1.0. The pair of clusters with the least loss of the log-likelihood will be agglomerated into a new cluster iteratively. The agglomeration procedure stops when the $BIC$ value reaches the minimum, which can lead to an optimal number of clusters. Starting from this interim mixture model, the overhead associated with the HAC algorithm is significantly alleviated [30]. In fact, the computational complexity is quadratic with the number of clusters in the interim mixture model.

Since the accurate calculation of the loss of the log-likelihood is rather time-consuming, it is approximated based on the assumption that $p_k \phi(x_k | \theta_k) >>$
\[ p_s \phi(x_k|\theta_s) \ (s \neq k) \] as \( x_k \) belongs to the \( k^{th} \) cluster \( C_k \). Then it can be approximated as

\[
L(\Phi) \approx \sum_{k=1}^{K} \sum_{x_i \in C_k} \log (p_k \phi(x_i|\theta_k))
\]

\[
\approx \sum_{k=1}^{K} N p_k \left[ \log p_k - \frac{\log |\Sigma_k|}{2} - \frac{D}{2} \log(2\pi) \right]
\]

\[
- \sum_{k=1}^{K} \sum_{x_i \in C_k} \left\{ \frac{1}{2} (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k) \right\}
\]

\[
\approx N \sum_{k=1}^{K} p_k \log p_k - \frac{1}{2} N \sum_{k=1}^{K} p_k \log |\Sigma_k| + \frac{ND}{2} \log(2\pi) - \frac{1}{2} ND,
\]

because

\[
\Sigma_k \approx \sum_{x_i \in C_k} \frac{(x_i - \mu_k)(x_i - \mu_k)^T}{N p_k}
\]

and

\[
\sum_{x_i \in C_k} \left\{ (x_i - \mu_k)^T \Sigma_k^{-1} (x_i - \mu_k) \right\}
\]

\[
= \sum_{x_i \in C_k} tr \left[ (x_i - \mu_k)(x_i - \mu_k)^T \Sigma_k^{-1} \right]
\]

\[
= tr \left( N p_k \Sigma_k \Sigma_k^{-1} \right) = N p_k D.
\]

Here \( \text{tr}(S) \) is the trace of the matrix \( S \). If clusters \( l \) and \( j \) are agglomerated into cluster \( r \), it can be derived that

\[
\Delta L(K) = L(\Phi_K) - L(\Phi_{K-1})
\]

\[
\approx N (p_l \log p_l + p_j \log p_j - p_r \log p_r)
\]

\[
- \frac{N}{2} (p_l \log |\Sigma_l| + p_j \log |\Sigma_j| - p_r \log |\Sigma_r|).
\]

As the number of free parameters for every cluster is fixed, say \( F \), \( BIC(\Phi_K) \) can be reformulated as

\[
\Delta BIC(K) = BIC(\Phi_K) - BIC(\Phi_{K-1})
\]

\[
\approx -2\Delta L(K) + F \log N.
\]
In order to minimize $BIC$ within the HAC agglomeration procedure, $\Delta BIC(K)$ should be greater than or equal to zero. That is

$$\Delta L(K) \leq \frac{F}{2} \log N. \quad (7.7)$$

Thus, a simple termination criterion is obtained for the HAC crossover operator to terminate with a good number of clusters.

Now let us derive the update formulae for the model parameters during the agglomeration procedure. We assume that, if clusters $j$ and $l$ agglomerate into cluster $r$, all their membership probabilities are agglomerated. So

$$u_r = \sum_{i=1}^{N} \frac{(t_{ij} + t_{il})}{N(p_j + p_l)} x_i = \frac{p_j \mu_j + p_l \mu_l}{p_j + p_l}$$

and

$$\Sigma_r = \sum_{i=1}^{N} \frac{(t_{ij} + t_{il})(x_i - \mu_r)(x_i - \mu_r)^T}{N(p_j + p_l)}$$

$$= \frac{\sum_{i=1}^{N} (t_{ij} + t_{il}) x_i x_i^T}{N(p_j + p_l)} - \mu_r \mu_r^T$$

$$= \frac{p_j \Sigma_j + p_l \Sigma_l}{p_j + p_l} + \frac{p_j p_l}{(p_j + p_l)^2} (\mu_j - \mu_l)(\mu_j - \mu_l)^T.$$  

Here $t_{ij}$ is the membership probability of $x_i$ belonging to the $j^{th}$ cluster. Since the HAC crossover operator does not need to access the data set again, it is suitable for processing large data sets. In addition, the computational complexity of the HAC crossover operator is $O((K_1 + K_2)^2)$ where $K_i (i = 1 \text{ or } 2)$ is the number of clusters of a parent individual.

### 7.1.3 Mutation Operators

The proposed GAXEM and GAEM have two mutation operators that introduce certain diversity into the population of chromosomes. These operators enable the algorithm to generate an individual far away from the old one, and they favor the global convergence of GAXEM and GAEM.
The first mutation operator modifies the parameters of the mixture model. It randomly selects a cluster, say $k$, and a data item, say $x_i$. Then the mean vector $\mu_k$ is moved toward $x_i$ with a small step. The covariance matrix $\Sigma_k$ is adjusted by the outer product of $x_i - \mu_k$. They are formulated as follows.

$$
\mu_k' = (1 - \alpha)\mu_k + \alpha x_i
$$
$$
\Sigma_k' = (1 - \beta)\Sigma_k + \beta(x_i - \mu_k)(x_i - \mu_k)^T
$$

where $\alpha$ and $\beta$ are two small random numbers.

The second mutation operator modifies the number of clusters $K$. It first generates a new number $K'$ around $K$. If $K' < K$, then it randomly selects $K'$ clusters to form a new chromosome. Otherwise, it selects some data items as the mean vectors and the identity matrices as the covariance matrices to form some additional cluster descriptions.
7.2 Experimental Results

We illustrate the performance of the proposed GAXEM and GAEM algorithms on various data sets and compare them with EnumEM and AutoClass [20]. All algorithms are implemented with MATLAB except that AutoClass is coded in the C programming language. The source program of AutoClass is obtained from the web site http://ic-www.arc.nasa.gov/ic/projects/bayes-group. All experiments are executed on Sun Ultra 5/270 Workstations. GAXEM and GAEM employ the elitism strategy, and they are terminated when the fitness of the best chromosome has not changed for 5 generations. To make a fair comparison with EnumEM and AutoClass, the population size is set as $\frac{5 \times N}{13}$. Here the number $\frac{1}{N}$ is the upper bound for the numbers of clusters, and the classical EM algorithm is invoked 5 times for every mixture model with a given number of clusters in EnumEM [30, 90]. Based on some preliminary experiments, the crossover and the mutation probability are set to 0.99 and 0.05 respectively. The two mutation and the two crossover operators are invoked with equal probability.

The first set of experiments is conducted on nine synthetic data sets. The means of Gaussian distribution components are located on the 2-dimensional grid and the covariance matrices are generated randomly around $0.1 \times I_2$, where $I_2$ is the 2*2 identity matrix. We draw about 200 data items from each cluster for the first 7 data sets and 250 for the last two data sets. These parameters are listed in Table 7.1 where $K$ indicates the optimal number of clusters and $N$ indicates the data size. The last data set dataSetI is depicted in Figure 7.3. The 3000 data items are indicated by dots, and a ‘+’ and its corresponding solid ellipse indicate the mean and the contour of a distribution component. The clusters in this data set are heavily overlapped.

The experimental results are summarized in Table 7.1. All results are based on 10 independent runs. We use three criteria to measure the performance. The classification accuracy and the successful trials of finding the optimal number of
Figure 7.4: Typical mixture models for dataSetI generated by (a). EnumEM, (b). AutoClass, (c). GAEM and (e). GAXEM. The minimal and the average $BIC$ values during a typical run of (d). GAEM and (f). GAXEM are depicted respectively.
Table 7.1: The experimental results on nine synthetic data of EnumEM, AutoClass, GAEM, and GAXEM. \( K \) and \( N \) indicate respectively the optimal number of clusters and the number of data items in a data set. ‘Accu’(\%) is the average classification accuracy value and ‘Suc’ indicates the number of successful runs within 10 runs. The time unit is seconds.

<table>
<thead>
<tr>
<th>Data set</th>
<th>( K )</th>
<th>( N )</th>
<th>EnumEM</th>
<th>AutoClass</th>
<th>GAEM</th>
<th>GAXEM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accu(%)</td>
<td>Suc</td>
<td>Time</td>
<td>Accu(%)</td>
<td>Suc</td>
<td>Time</td>
</tr>
<tr>
<td>dataSetA</td>
<td>4</td>
<td>800</td>
<td>56.8</td>
<td>8</td>
<td>1799</td>
<td>63.5</td>
</tr>
<tr>
<td>dataSetB</td>
<td>5</td>
<td>1000</td>
<td>51.4</td>
<td>14</td>
<td>2951</td>
<td>53.7</td>
</tr>
<tr>
<td>dataSetC</td>
<td>6</td>
<td>1200</td>
<td>42.6</td>
<td>5</td>
<td>2194</td>
<td>46.4</td>
</tr>
<tr>
<td>dataSetD</td>
<td>7</td>
<td>1400</td>
<td>42.8</td>
<td>2</td>
<td>3135</td>
<td>56.6</td>
</tr>
<tr>
<td>dataSetE</td>
<td>8</td>
<td>1600</td>
<td>62.0</td>
<td>4</td>
<td>3224</td>
<td>63.7</td>
</tr>
<tr>
<td>dataSetF</td>
<td>9</td>
<td>1800</td>
<td>53.9</td>
<td>3</td>
<td>3318</td>
<td>60.2</td>
</tr>
<tr>
<td>dataSetG</td>
<td>10</td>
<td>2000</td>
<td>54.0</td>
<td>3</td>
<td>4369</td>
<td>55.2</td>
</tr>
<tr>
<td>dataSetH</td>
<td>11</td>
<td>2500</td>
<td>45.2</td>
<td>2</td>
<td>8570</td>
<td>47.1</td>
</tr>
<tr>
<td>dataSetI</td>
<td>12</td>
<td>3000</td>
<td>37.5</td>
<td>2</td>
<td>9149</td>
<td>43.6</td>
</tr>
<tr>
<td>Average</td>
<td>49.6</td>
<td>3.8</td>
<td>4259</td>
<td>54.7</td>
<td>7.1</td>
<td>1737</td>
</tr>
</tbody>
</table>

Clusters examine the accuracy issue, while the execution time measures the computation resource required. In Table 7.1, ‘Suc’ indicates the number of successful runs of finding the optimal number of clusters within 10 runs. For example, for the data set ‘dataSetI’ depicted in Figure 7.3, GAXEM succeeds in determining 12 clusters 7 times within 10 runs. However, EnumEM, AutoClass and GAEM succeed 2, 4, and 3 times within 10 runs, respectively. So, GAXEM succeeds more frequently in determining the optimal number of clusters than its counterparts. The typical mixture models generated by EnumEM, AutoClass, GAEM, and GAXEM are illustrated in Figures 7.4(a), (b), (c) and (e) respectively. In the figures, a Gaussian distribution is indicated by a ‘+’ and its associated ellipse. In Figure 7.4(e), we can see that GAXEM generates a mixture model similar to the original one as shown in Figure 7.3. The fitness values during a typical run of GAEM and GAXEM are illustrated respectively in Figures 7.4(d) and 7.4(f).

In the figures, a dashed line indicates the average $BIC$ value and a solid line indicates the minimal $BIC$ value during the runs. Though the $BIC$ value of GAEM and GAXEM are very close, GAEM generates a mixture model with 11 clusters and GAXEM finds the correct number of clusters.
GAXEM can determine the optimal number of clusters more frequently than the others for all data sets except for dataSetB and dataSetF. On average, EnumEM, GAEM, AutoClass, and GAXEM succeed respectively 3.8, 6.2, 7.1, and 8.3 times within 10 runs. GAXEM outperforms the other three algorithms. A similar situation can be observed on the classification accuracy. It is defined to measure the match between two classifications $C$ and $C'$ by

$$\text{Accuracy}(C, C') = 1 - \frac{E(C, C') + E(C', C)}{2}$$

and $E(C, C') = \sum_{k=1}^{K} p_k \left[ -\sum_{i=1}^{K'} q_{ki'} \log q_{ki'} \right]$, where $q_{ki'}$ is the ratio of data items in cluster $k$ of classification $C$ assigned to cluster $i'$ of classification $C'$. The definition does not require two classifications to have the same number of clusters. The classification accuracy value reaches the maximum 1.0 as two classifications are identical. We use the classification generated by the original mixture model as the reference. Since the data sets are ill-separated, the classification accuracy values in Table 7.1 with respect to the original classification are hard to approach one. GAXEM generates better classifications with higher accuracy values than EnumEM, AutoClass, and GAEM for all the data sets except for dataSetB and dataSetF. GAEM performs better than EnumEM for all the data sets except for dataSetH. The average classification accuracy values of GAXEM, GAEM, AutoClass, and EnumEM are 57.6%, 54.8%, 54.7%, and 49.6%, respectively. In summary, the solutions obtained by GAXEM are better than those obtained by the other three algorithms. GAXEM performs better than GAEM, which confirms the significance of the proposed HAC crossover operator.

On the other hand, though GAXEM and GAEM require longer computation time, their computation times grow in a similar way as those of EnumEM. Their computation times is within 10 times of that of AutoClass, which runs faster partially because it is coded in the C programming language [20].

Our second set of experiments is conducted on five real-life data sets from the UCI machine learning repository. Its web site is www.sgi.com/Technology/mlc/db.
The experimental results are summarized in Table 7.2 where the ‘Attributes’ column indicates the attributes used. For example, for the ‘thyroid’ data set, we employ 5 attributes. GAXEM determines 3 clusters correctly 8 times within 10 runs. AutoClass and EnumEM succeed 7 and 6 times respectively. Similar situation may be found for the other data sets. It can be observed from the table that GAXEM can determine the optimal number of clusters in most experiments. On average, GAXEM can succeed 8.6 times within 10 independent runs. AutoClass and EnumEM succeed respectively 4.4 and 6.4 times. The comparison on another measurement, $BIC$, is consistent. Normally, GAXEM generates better solutions with lower $BIC$ values than EnumEM and AutoClass for all 5 data sets. AutoClass is apt to generate more clusters than the optimal. Especially, it fails to detect the five clusters in the data set ‘sleep’ within 10 runs. This point may be caused by the ‘Occam Factor’ employed in AutoClass [20] which differs from the $BIC$ values in GAXEM and EnumEM for measuring model suitability. On the other hand, though GAXEM normally needs longer execution time, its average execution times are within three times of those of EnumEM and AutoClass.

In summary, GAXEM can determine the optimal number of clusters more frequently than EnumEM, AutoClass, and GAEM using longer execution time. GAEM, similar to AutoClass, performs better than EnumEM. GAXEM outperforms GAEM which shows the important role of the proposed HAC crossover operators.

### 7.3 Scalable Model Selection

The genetic-guided model-based clustering algorithms can frequently determine the appropriate number of clusters of data sets. However, the execution time increases very quickly with the data size. In fact, All of EnumEM, GAXEM, GAEM, and AutoClass needs to invoke the classical EM algorithm multiple times. According to our experiments in Chapters 5 and 6, the classical EM algorithm
Table 7.2: The experimental results of EnumEM, AutoClass and GAXEM on five real world data sets. \( K \) and \( N \) indicate the optimal number of clusters and the number of data items in a data set, respectively. ‘Attributes’ indicates the attributes used in the experiments. ‘\( BIC \)’ indicates the average \( BIC \) values, ‘\( \text{Accu}(\%) \)’ is the average classification accuracy value and ‘Suc’ indicates the successful runs within 10 runs. The unit for the execution time is second.

<table>
<thead>
<tr>
<th>data set</th>
<th>name</th>
<th>( K )</th>
<th>( N )</th>
<th>Attributes</th>
<th>( BIC )</th>
<th>( \text{Accu}(%) )</th>
<th>Time</th>
<th>Suc</th>
<th>( BIC )</th>
<th>( \text{Accu}(%) )</th>
<th>Time</th>
<th>Suc</th>
<th>( BIC )</th>
<th>( \text{Accu}(%) )</th>
<th>Time</th>
<th>Suc</th>
</tr>
</thead>
<tbody>
<tr>
<td>diabetes</td>
<td>3</td>
<td>145</td>
<td>1,2,3</td>
<td>4762</td>
<td>60.4</td>
<td>81</td>
<td>9</td>
<td>4766</td>
<td>60.2</td>
<td>68</td>
<td>9</td>
<td>4770</td>
<td>60.7</td>
<td>113</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>thyroid</td>
<td>3</td>
<td>215</td>
<td>1,2,3,4,5</td>
<td>4948</td>
<td>74.0</td>
<td>104</td>
<td>6</td>
<td>4921</td>
<td>79.7</td>
<td>108</td>
<td>7</td>
<td>4810</td>
<td>83.0</td>
<td>203</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>iris</td>
<td>3</td>
<td>150</td>
<td>1,3,4</td>
<td>514</td>
<td>90.6</td>
<td>88</td>
<td>7</td>
<td>573</td>
<td>62.1</td>
<td>87</td>
<td>2</td>
<td>510</td>
<td>91.2</td>
<td>124</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>liver</td>
<td>2</td>
<td>345</td>
<td>1,2,3,4</td>
<td>1059</td>
<td>38.5</td>
<td>485</td>
<td>6</td>
<td>1223</td>
<td>36.9</td>
<td>264</td>
<td>4</td>
<td>1009</td>
<td>40.8</td>
<td>570</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>sleep</td>
<td>5</td>
<td>2500</td>
<td>5,7,8,10</td>
<td>38832</td>
<td>45.7</td>
<td>3045</td>
<td>4</td>
<td>42233</td>
<td>38.4</td>
<td>3481</td>
<td>0</td>
<td>37994</td>
<td>50.2</td>
<td>8793</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>10043</td>
<td>61.8</td>
<td>761</td>
<td>6.4</td>
<td>10743</td>
<td>55.5</td>
<td>802</td>
<td>4.4</td>
<td>9819</td>
<td>65.2</td>
<td>1961</td>
<td>8.6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

is not very suitable for large data sets. So, we need new techniques for scalable model selection that determines the number of clusters of large data sets.

One of the scalable model selection methods may be derived from the framework discussed in Chapter 3. Our basic idea, similar to the scalable clustering algorithms in Chapters 5 and 6, is to work on summary statistics directly. A large data set is first summarized into a set of compact summary statistics. This set of summary statistics is easily stored in the given main memory. Then the in-memory clustering algorithms can generate mixture models from the summary statistics effectively and efficiently. In order to find the appropriate number of clusters, some global search algorithms are used to guide the in-memory clustering algorithms to find the best mixture models. We have discussed two kinds of summary statistics for subclusters in Chapter 3. For simplicity, we take clustering features as summary statistics to illustrate our idea in the following discussion. The basic idea is readily applicable to data summaries.

Our suggested algorithm is called GAEMACF that is a combination of GA with EMACF. Broadly speaking, the global algorithm GA guides the local search algorithm EMACF during the search for the best mixture model. GA provides some global information to focus EMACF on some promising area. GA generates...
some good mixture models to be the initialization of EMACF. EMACF refines mixture models only based on the clustering features. As shown in Chapter 5, EMACF generates local optimal mixture models. Their log-likelihood values are employed to calculate a certain criterion, say, the $BIC$ value. Then we may measure the model suitability accurately. Moreover, since EMACF only reads the set of clustering features instead of the whole data set, the local search performs very quickly.

Now we discuss the genetic operators in GAXEM. As mentioned in Section 7.1, the HAC crossover operator terminates the agglomeration procedure with a low $BIC$ value and generates a good mixture model from the two old mixture models. So, the HAC crossover operator provides an interesting interaction between different solutions. In addition, its computational complexity is quadratic with respect to the total number of clusters in the two old mixture models. It does not has to read through the whole data set again. In other words, the HAC crossover operator is not dependent on the data size. Thus, it is directly applicable to large data sets. Similarly, the other genetic operators described in Section 7.1 are also applicable in GAEMACF.

### 7.4 Chapter Summary

In this chapter, we have discussed the genetic-guided model-based clustering algorithms that determine the number of clusters automatically. We have established several interesting genetic operators. The experimental results on both the synthetic and the real world data sets have substantiated that the proposed GAXEM algorithm can frequently determine the appropriate number of clusters. We have also discussed how to determine the number of clusters in large data sets by combining GAXEM with the techniques established in Chapters 5 and 6.
Chapter 8

Discussion and Conclusions

In this chapter, we first discuss two additional possible applications of our proposed scalable model-based clustering systems. They are used to detect outliers and optimize database query in Section 8.1. The contributions of the thesis are summarized in Section 8.2. Finally, we discuss the limitations of our scalable model-based clustering systems and give some possible future directions in Section 8.3. The future work mainly aims to further enhance the work we have done in the thesis.

8.1 Applications

Besides the cluster analysis for large databases discussed in the previous chapters, our established scalable model-based clustering systems can be employed to conduct some other interesting applications. We discuss its application to the outlier detection in Section 8.1.1, and its application to the database query optimization in Section 8.1.2.

8.1.1 Outlier Detection

The outlier (or noise) detection or analysis is one of the interesting data mining tasks. The data items that are grossly different from or inconsistent with the remaining set of data are called outliers. These outliers can be of particular
interest in practice. In the case of fraud detection, outliers may indicate fraudulent activity, say, unusual usage of credit cards or telecommunication services. The outlier detection is also useful in customized marketing for identifying the spending behavior of customers with extremely low or extremely high incomes.

The mixture models generated by our model-based clustering algorithms can be applied to detect outliers directly. We discuss and illustrate two interesting methods.

The first method is to associate outliers with some special clusters. In other words, some component distributions describe the inherent clusters of a data set,
and some other component distributions describe the outliers of the data set. We further explain this method with a mixture model generated by EMADS. We plot the California housing data in the Latitude-Longitude space as shown in Figure 8.1. In the figure, the two attributes are scaled into \([0, 3]\). A dot or a ‘x’ indicates a data item. The seven Gaussian distributions are indicated by seven ellipses and their associated ‘o’. Compared with other Gaussian distributions, the Gaussian distribution indicated by the biggest ellipse is unusual. It overlaps five other Gaussian distributions. So, we regard the data items associated with this Gaussian distribution as the outliers, as indicated by ‘x’ in the figure. It is obvious that these data items distribute in the relatively sparse areas. These sparse areas are obviously distinguishable from the six areas indicated by the other six Gaussian distributions. Thus, the inherent clusters and outlier are clearly identified by the generated mixture model.

The second method is based on the probability of a generated mixture model. A data item is detected as an outlier if its probability under the mixture model is lower than a small threshold. In other words, it is not strongly associated with any cluster. We use a mixture model generated by EMACF to illustrate this method. A typical mixture model generated by EMACF is depicted in Figure 8.2, where six ellipses and their associated ‘o’ indicate the six Gaussian distributions. This data set has been used to examine the performance of EMACF in Chapter 5. A data item, if its probability under the mixture model is less than 0.001, is identified as an outlier. It is indicated by an ‘x’ in the figure. The other data items are indicated by the dots. It is clearly observed that the outliers are normally in sparse areas and are far away from any cluster. In other words, these outliers do not belong to the six clusters. So, we have successfully identified the outliers in the data set.
Figure 8.2: The six clusters and outlier in a data set. An outlier is indicated by an ‘x’. Six clusters are indicated by six ellipses and their associated ‘o’.

### 8.1.2 Query Optimization

Our clustering algorithms can generate mixture models with local maximum likelihood for a large data set. The maximum likelihood estimation can be regarded as a good density estimation of the data set. It is known that any distribution can be approximated accurately with a mixture model containing a sufficient number of components [64]. So, the mixture model generated by our model-based clustering algorithms can be employed to estimate the density of large data sets effectively and efficiently. Here we take the aggregate query optimization as an example to show an application of the generated mixture models.

In the following, we take account of four types of aggregate queries: count, sum, standard deviation, and fuzzy average. These queries are frequently required
in decision making. The standard language for relational database management systems, SQL, supports the first three queries. However, the query processing procedure often needs to scan the whole database. It becomes quite time-consuming when the database is very large [77]. In addition, the standard SQL does not support fuzzy queries, which occur usually in our daily life. If we generate efficiently the mixture model associated with a query, or we reuse the generated mixture model, we can get approximate query results very quickly. The prompt and approximate results are quite useful. They facilitate users to get a high-level view of the database quickly and provide some guidance for further exploration of the database.

We illustrate this interesting application with a census database. The census database contains columns like ID, Age, Income, and Education Level. For simplicity, we use three metric variables $x_1$, $x_2$ and $x_3$ to describe the last three attributes, respectively. Our scalable model-based clustering systems may generate a good Gaussian mixture model, say $\Psi$, for three attributes efficiently. We assume there are totally $N$ rows in the database. We list the approximate answers for the four aggregate queries using the density function $p(x|\Phi)$.

**Count:** How many people is older than 30 and younger than 40 in the census database? The standard SQL uses a SQL select statement like

```
select count(ID)
from census
where Age between 30 and 40;
```

The approximate answer based on the mixture model $\Phi$ is simply

$$answer = N \times \int_{30}^{40} \int_{0}^{+\infty} \int_{0}^{+\infty} p(x|\Phi)dx_1dx_2dx_3. \quad (8.1)$$

**Sum:** What is the total income of people aged between 30 and 40? The answer from SQL is
select sum(income)
from census
where Age between 30 and 40;

The answer can be simply approximated as:

$$answer = N \times \int_{30}^{40} \int_{0}^{+\infty} \int_{0}^{+\infty} x_2 \times p(x|\Phi)dx_1dx_2dx_3. \quad (8.2)$$

**Standard Deviation:** What is the standard deviation of income for people aged between 30 and 40? Two SQL statements are required:

select sum(income*income)
from census
where Age between 30 and 40;

and

select ave(income)
from census
where Age between 30 and 40;

We use $a_1$ and $a_2$ to denote the results of the above two queries respectively, then the standard deviation is $\sqrt{\frac{a_1}{N} - a_2^2}$. The approximate answer from the mixture model is

$$answer = \sqrt{ \int_{30}^{40} \int_{0}^{+\infty} \int_{0}^{+\infty} x_2^2 \times p(x|\Phi)dx_1dx_2dx_3 - a^2}$$

where $a = \int_{30}^{40} \int_{0}^{+\infty} \int_{0}^{+\infty} x_2 \times p(x|\Phi)dx_1dx_2dx_3 \quad (8.3)$

**Fuzzy Average:** What is the average income of the old people? Since the query contains a fuzzy item, it is difficult to write out a SQL select statement even if a fuzzy membership function $f(x_1)$ is given. The fuzzy membership
function $f(x_1)$ measures the fuzzy item ‘old’. However, we may get a good answer from the mixture model

$$\text{answer} = \frac{\int_0^{+\infty} \int_0^{+\infty} \int_0^{+\infty} f(x_1) \times x_2 \times p(x|\Phi) dx_1 dx_2 dx_3}{\int_0^{+\infty} \int_0^{+\infty} \int_0^{+\infty} f(x_1) \times p(x|\Phi) dx_1 dx_2 dx_3}$$  \hspace{1cm} (8.4)

A Gaussian mixture model is a linear combination of several Gaussian distributions. The above integrals are sums of several integrals on some Gaussian distributions which are computationally cheap. So, the execution time of the approximate results is quite short. For these two methods, as reported in [77], the execution time ratios range from 1,556 to 35,000 for the first two kinds queries. The data sizes range from 299,281 to 835,600. Meanwhile, the approximate results have less than 5% errors in comparison with the actual results. The execution time comparison between the two methods becomes more significant when such queries are repeatedly frequently used. The queries including fuzzy items are quite common in our daily life. It is reasonable to generate an approximate result for a fuzzy query. On the other hand, to the best of our knowledge, there is no direct support from the standard SQL.

### 8.2 Contributions

With the rapid development of information technology, large databases become ubiquitous. A number of challenges have arisen on how to manage and how to explore the large databases. The cluster analysis of large databases becomes one of the main challenges in the data mining community. The real world problems urge for some scalable clustering systems. These systems enable cluster analysis of large databases with restricted main memory and linear computation time with respect to the data size.

In response to the challenge of scalable cluster analysis, we have proposed
some promising model-based techniques to work on summary statistics of sub-clusters of data items. Instead of handling each individual data item separately, a subcluster of similar data items are handled as one data unit. On one hand, we can dramatically reduce the amount of main memory by only storing the summary statistics. On the other hand, the computation time increases slowly with the number of data items. In the data summarization procedure, each data item can be efficiently assigned to its associated subcluster using some indexing techniques. In the mixture model generation procedure, the in-memory model-based clustering algorithms can handle the small set of summary statistics efficiently. Furthermore, since the data distribution within each subcluster has been employed in our model-based clustering algorithms, our clustering systems have achieved high clustering accuracy.

The research challenges of clustering in data mining have been discussed in Section 1.2.1. Besides the scalability, we have also tried to meet other five challenges as described in various chapters in the thesis. A quick summary is given below.

1. **Discovery of clusters with arbitrary shape:** For different kinds of data attributes, we have developed two kinds of summary statistics. We have used a clustering feature to describe a subcluster of data items if the data attributes are statistically independent. Then we have established EMACF (the Expectation Maximization Algorithm for Clustering Features) to generate mixture models from clustering features directly. Two clustering systems, the gEMACF and the bEMACF algorithms, have been developed to discover the ellipse-like clusters in Chapter 5. However, these ellipses are only allowed to expand along with the attributes. When the data attributes are correlated, we have used a data summary to indicate a subcluster of data items and developed EMADS (the EM Algorithm for Data Summaries). Ellipse-like clusters with arbitrary orientation can be detected by our clustering systems based on EMADS, i.e., the gEMADS
and the bEMADS algorithms in Section 6.2.

2. **Sound theoretic support:** Our proposed model-based clustering algorithms are derived from the general EM algorithm. All theoretic results on the general EM algorithm are applicable to our new model-based clustering algorithms. We have proved the convergence theorems of EMACF and EMADS in Sections 5.1 and 6.1, respectively.

3. **Insensitivity to the order of input data:** The data input order may impact the data summarization procedures as discussed in Chapter 4, and they may generate some different summary statistics. However, both EMACF and EMADS scan the whole set of summary statistics per iteration. They are insensitive to the input order of the summary statistics, and thus they are not very sensitive to the data summarization procedures. Finally, the order of input data slightly influences the clustering results. The sensitivity studies of EMACF and EMADS have been given in Sections 5.2.5 and 6.2.3, respectively.

4. **Minimal requirements for domain knowledge to determine input parameters:** In Chapter 7, we have discussed some techniques to determine the number of clusters. The proposed GAXEM algorithm can guide our clustering systems to determine the optimal number of clusters effectively.

5. **Ability to deal with noisy data:** Our clustering systems are able to handle noisy data robustly. These noisy data can be detected as early as at the data summarization phase. Several robust data summarization procedures have been discussed in Chapter 4. Some simple examples for outlier detection have been illustrated in Section 8.1.1.

The top-level contribution of this thesis is the substantiation of the claim stated in Chapter 1, which is, “*Working on summary statistics of subclusters is a*
sound and effective strategy to scale-up model-based clustering techniques.” This claim is substantiated by the design, implementation, and evaluation of our four scalable clustering systems. At the next level of detail, the following specific contributions have been made:

- We have presented a multi-phase framework for scalable model-based cluster analysis. It first roughly partitions a data set into subclusters. Each subcluster of data items are summarized into summary statistics. Then it generates accurate clusters from the summary statistics directly. Finally, a sophisticated controller may manipulate the whole clustering procedure. These functionalities are fulfilled with three modules. They are the data summarization, the in-memory cluster analysis, and the optional controller modules. (Section 3.2)

- We have established two scalable model-based clustering systems for subclusters represented in clustering features. These systems are suitable for the data sets with statistically independent attributes. The detailed work is listed as follows.

  - We have derived a new model-based clustering algorithm, EMACF. It can generate mixture models from the clustering features directly and effectively. (Section 5.1.2)

  - We have conducted some theoretical analyses. The convergence theorem guarantees that EMACF converges to local maxima. Its computational complexity is linear with the number of clustering features. (Sections 5.1.3 and 5.1.4)

  - We have established two scalable model-based clustering systems, the gEMACF and the bEMACF algorithms by combining EMACF with the adaptive grid-based data summarization and the BIRCH’s data summarization procedures, respectively. Both the gEMACF and the
bEMACF algorithms can generate clustering results with no or little loss of accuracy and run one or two orders of magnitude faster than the classical EM algorithm on synthetic data sets. They can run faster and generate more accurate results than the sampling EM algorithm. They can generate more accurate clustering results with slightly longer time than the scalable clustering systems based on the ExEM algorithm. Similar comparison results have been observed when the bEMACF algorithm is applied to three real-world data sets. (Sections 4.1, 5.2.3, 5.2.4, 5.3.2, 5.3.3, and 5.3.4)

- We have established two scalable model-based clustering systems for subclusters represented in data summaries. These systems can take account of the correlations among attributes. The detailed work is listed as follows.

  - We have simplified the data summary representation scheme based on a mathematically sound theorem. (Section 6.1.1)
  - Using the simplified data summaries, we have derived EMADS from the general EM algorithm. It can generate mixture models from the data summaries directly effectively. (Section 6.1.2)
  - We have conducted some theoretical analyses. The convergence theorem guarantees that EMADS converges to local maxima. Its computational complexity is linear with the number of data summaries. (Sections 6.1.3 and 6.1.4)
  - Combining EMADS with the adaptive grid-based data summarization and the BIRCH’s data summarization procedures, we have established two clustering systems, the gEMADS and the bEMADS algorithms, respectively. The bEMADS algorithm can run one or two orders of magnitude faster than the classical EM algorithm with no or little loss of clustering accuracy on both synthetic and
real world data sets. It outperforms the sampling EM algorithm in terms of both the execution time and the clustering accuracy. It generated much more accurate clustering results than the bE-MACF algorithm with slightly longer time. (Sections 4.1, 6.2.3, 6.2.4, 6.2.5, and 6.2.6)

- We have proposed an Expanding Self-Organizing Map (ESOM) and Integrated Self-Organizing Map (ISOM) for better data summarization and projection. The ESOM and the ISOM can generate the most accurate solutions for the travelling salesman problem in the neural network literature so far. (Sections 4.2.2, 4.2.3, A.3.2, and B.3.2)

- We have combined genetic algorithms with the model-based clustering techniques and established the GAXEM algorithm. The GAXEM algorithm can determine the optimal number of clusters more frequently than two existing algorithms with longer execution time. (Sections 7.1 and 7.2)

8.3 Limitations and Future Work

Our work can be improved or extended in a number of ways. In the following, eight suggestions are made. They are in fact suggestions of two different scopes. The first five are relatively minor enhancements or extensions to the current work. The last four are more major undertakings.

Robust cluster analysis: Our model-based clustering systems handle noisy data at the data summarization phase. For example, we have deleted the noisy data using a mechanism of the BIRCH’s data summarization procedure. We need a comprehensive set of experiments to examine the robustness of the proposed clustering systems to noise. The study along this way may
help us to establish some robust scalable clustering systems. Incorporating the outlier detection property of the model-based clustering techniques, these robust clustering systems can then be applicable to outlier analysis of large data sets. Another interesting issue is to examine the robustness of our systems with respect to complicate data sets, say, the data sets with overlapping clusters.

**Database query optimization:** We have shown the feasibility of applying our scalable model-based clustering techniques to accelerate database aggregate queries. Especially, we have given a kind of fuzzy queries which are suitable for our model-based clustering algorithms. To the best of our knowledge, they are not directly supported by the standard SQL. Once experimental results substantiate that the approximate results from our clustering algorithms are reasonably acceptable, we may embed our clustering algorithms into the query processing engine. It finally enables the relational database management systems to handle the fuzzy queries.

**Scalable determining the number of clusters:** It is a challenging task for us to automatically determine the optimal number of clusters in a large database. The information theoretic criteria, e.g., the Bayesian Information Criterion, have been successfully used in small data sets. We still require some statistically sound supports before applying them to large databases. Otherwise, we have to derive some new criteria to help us to determine the optimal number of clusters.

**Scalable clustering for high dimensional databases:** Applications in various domains often lead to very high-dimensional data; the dimension of the data is in hundreds or thousands, for example in text/web mining and bioinformatics. In addition to the high dimensionality, these data sets are often sparse. Clustering such high-dimensional data sets is a contemporary challenge. Successful algorithms must avoid the curse of dimensionality but
at the same time should be computationally efficient [1, 44]. How to scale-up model-based clustering techniques for high-dimensional databases is also of future research interest.

**Better interaction among different modules:** In the experimental study of the thesis, we have only given some heuristic methods to determine the granularity of subclusters. In order to facilitate end users, we need some more sophisticated methods to adjust the data summarization procedures. Then, the in-memory clustering algorithm, say, EMACF and EMADS, can cooperate seamlessly with the data summarization procedures and generate more accurate clustering results.

**New summary statistics:** Two kinds of summary statistics, clustering features and data summaries, have been employed to describe the subclusters. Intuitively, a subcluster of data items may be described more accurately if more information is included. For example, if we store the minimal and the maximal values within a subcluster into the summary statistics, we want to examine whether the model-based clustering algorithms can achieve higher clustering accuracy. Besides the zeroth, the first, and the second moments, there are some higher-order statistics that may be generated from a subcluster of data items. So, another consideration is whether it is worthwhile to embed some higher-order statistics into the summary statistics for higher clustering accuracy.

**On-line learning mode:** Both EMACF and EMADS are iterative algorithms. As discussed in Section 6.3, the on-line learning mode is also applicable to EMACF and EMADS. The mixture model is adjusted according to summary statistics one by one, instead of the whole set of summary statistics. There are two main concerns. The first one is how to set the learning rate. The second one is how to reduce the sensitivity to the input order of the summary statistics.
Understandable cluster analysis: The clustering results generated by our model-based clustering algorithms are statistically sound. But it is not so easy to interpret and understand the cluster structure in a high-dimensional space since each cluster is described by a high-dimensional distribution. Obviously, only the end user can determine whether the clustering results are useful or not. Therefore, the interpretability and understandability become a challenging issue of the model-based clustering systems.

We may transform a high-dimensional distribution into a rule which human being is easy to understand and evaluate. This kind of transformation becomes difficult when the correlations between attributes exist.

Another method is to use some visualization techniques. It uses graphical and numerical tools to illustrate the hidden patterns or the generated clusters. It is a more effective approach for understanding and communication than the use of common numerical tools alone. Combining the model-based clustering techniques and the self-organization map visualization techniques, we may establish some new visual model-based clustering systems. They may have good interaction capability and knowledge transfer, and finally facilitate the end users.

Mixed attributes: The underlying principle of our scalable model-based clustering systems is applicable to data set with other types of attributes. We may use multinomial distributions for data with nominal attributes and Markov chains for time series. For data sets with mixed attributes, we may combine several distributions together for a single cluster. The first major challenge is to specify an appropriate type of summary statistics for a subcluster of similar data items. The summary statistics have to be very compact to reduce the main memory requirement. They also should sufficiently describe a subcluster so as to reduce the loss of clustering accuracy.
The second challenge is to establish some new model-based clustering algorithms to generate mixture models from the summary statistics directly and effectively. The third challenge is how to select a proper distribution for a single cluster. Even for a numeric attribute, we may use such a distribution as Gaussian, log-normal or Gamma. This is another type of model selection problem. All these are some of our future research interests.
Appendix A

ESOM for TSP

In Section 4.2.2, we have developed the Expanding Self-Organization Map (ESOM) for data summarization and projection. In the appendix, we apply the ESOM to the Traveling Salesman Problem (TSP) by incorporating SOM’s topological order-preserving property and the convex-hull property of the TSP. The ESOM may acquire both the local optimality (i.e., the topological order-preserving property) and the global optimality (i.e., the convex-hull property) of the TSP, and hence it can yield very near-optimal solutions. A series of experiments are conducted on both synthetic and benchmark TSPs. Simulation results demonstrate the superiority of the ESOM over several typical SOM-like networks such as the SOM network developed by Budinich [14], the convex elastic net [3], the KNIES algorithms [5], as well as the simulated annealing approach [52]. This indicates that the ESOM algorithm is one of the best neural networks in the literature.

We review the SOM for TSP in Section A.1. Then we rewrite the ESOM algorithm for the TSP in Section A.2. Finally, comprehensive experimental and comparison results are presented in Section A.3.

A.1 TSP and SOM

The TSP is one of the typical combinatorial optimization problems. It can be stated as a search for the shortest closed tour that visits each city once and only
Figure A.1: A Schematic view of a SOM for the TSP. In this appendix, \( p = 2 \).

The SOM-like neural networks, due to their acceptable computational complexity and promising performance, have attracted a large amount of research to explore and enhance the capability of handling the TSP, and they have generated some encouraging results [3, 14, 47, 59, 80]. When applied to the TSP, a SOM can be structured as in Figure A.1. A ring of output neurons serve to characterize the feature map denoted by \( 1, 2, \ldots, M \). The input neurons describe the coordinates of the given cities and are fully connected to every output neuron. If the state of input neurons at time \( t \) is the vector \( \mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_p(t)]^T \in \mathbb{R}^p \), where \( p \) is the number of input neurons, then the connection weights between the \( j^{th} \) output neuron to the input neurons comprise the vector \( \mathbf{w}_j(t) = [w_{j1}(t), w_{j2}(t), \ldots, w_{jp}(t)]^T \in \mathbb{R}^p \) \((1 \leq j \leq M)\). So the output
neurons have two topological domains. One lies in the ring of the output neurons to reflect a linear order of visiting the cities. The other lies in the $p$-dimensional space where the connection weight vector of each output neuron is viewed as its coordinates. The SOM may establish a good mapping from $p$-dimensional space to 1-dimensional ring.

Upon repeated presentations of the input data, the synaptic weight vectors in a SOM tend to follow the distribution of the input data. Therefore the SOM forms a topological order-preserving feature map of the input space in the sense that the neurons adjacent in the ring will tend to have similar weight vectors. Through this topologically ordered map, the SOM may generate good solutions of a TSP. So the solutions can be regarded as the consequence of the SOM that learns the local optimality of the TSP. That is, each subpath is locally shortest. However, it is not necessarily true that this local optimality exists for all TSPs and can be learned by the SOM. So there is no guarantee for the SOM to always yield an optimal tour of a TSP.

A.2 The ESOM for the TSP

To improve the performance, we apply the ESOM algorithm discussed in Section 4.2.2 to the TSP. It can learn the local optimality and certain global optimality simultaneously. Here by the global optimality we mean a property that any optimal tour of the TSP must have. The convex hull property is an example.

The convex hull for a TSP is the largest convex polygon whose vertices are cities of the TSP. The convex hull property says that, for any optimal tour of the TSP, the cities located on the convex hull must be visited in the same sequence as they appear on the convex hull [3, 56]. Let us take a 9-city TSP in Figure A.2(a) as an example, where the dotted lines constitute its convex hull. The property says that, since cities 3, 9, 5 and 7 appear on the convex hull, they should appear in clockwise or anti-clockwise order in the optimal tour, as in the one indicated
by solid lines in Figure A.2(a). On the contrary, the tour in Figure A.2(b) does not possess the property, so it is not an optimal one, as also indicated by its intersection. The property implies that, to get better tours, we ought to make better use of the cities located on the convex hull.

Our idea is to use the expanding force in the ESOM’s learning rule to achieve the convex hull property. According to the learning rule defined in Eq. (4.7), the excited neuron is drawn closer to the input city and pushed away from the center of cities. It is worth noting that pushing a neuron away from the center of cities can be viewed as approaching the convex hull. This learning procedure is clearly illustrated in Figure 4.6(b) on Page 45. First, the exited neuron is drawn closer to the input city, and then it is pushed towards the convex hull. The former adaptation, together with the cooperative adaptation of neighbor neurons, will gradually discover the topological neighborhood feature of the input data, as widely believed in the SOM research community. In a similar way, the latter adaptation, also together with the cooperative adaptation, can approach the convex hull property of the TSP. Therefore, our ESOM can acquire the convex hull property gradually as the topological neighborhood relationships among cities are
being inspected and preserved. Accordingly, nearer optimal solutions can be expected. In order to learn the convex hull property of optimal tours appropriately, it is also necessary for the latter adaptation to reflect the convex hull property in a single learning iteration. In other words, the expanding coefficient will reflect the convex hull to some degrees. We explain this point after the description of the ESOM algorithm below.

**Algorithm A.1 (The ESOM for the TSP)**

1. Map all the city coordinates \((x'_{k1}, x'_{k2})^T (i = 1, \ldots, n)\) into a circle \(C_R\) centered at the origin with radius \(R(\leq 1)\). The new coordinates are denoted by \((x_{k1}, x_{k2})^T\) below. The center of cities is mapped to the origin (the center of the circle).

2. Randomly set the weight vectors \(w_j(0) (j = 1, \ldots, M)\) within \(C_R\), and set \(t\) be 0.

3. Select a city at random, say \(x_k(t) = (x_{k1}(t), x_{k2}(t))^T\), and feed it to the input neurons.

4. Find the winning neuron, say \(m(t)\), nearest to \(x_k(t)\) according to the Euclidean metric.

5. Train neuron \(m(t)\) and its neighbors up to the width \(\sigma(t)\) with the following formula:

   \[
   w_j(t + 1) = c_j(t) \times [w_j(t) + \alpha_j(t) (x_k(t) - w_j(t))] \tag{A.1}
   \]

   where \(j = 0, 1, \ldots, M\), \(\alpha_j(t) \in [0, 1]\) is the learning rate specified by

   \[
   \alpha_j(t) = \eta(t) h_{j,m(t)} = \max \left(0, \eta(t) \left(1 - \frac{d_{j,m(t)}}{\sigma(t) + 1}\right)\right) \tag{A.2}
   \]

   and \(c_j(t)\) is the expanding coefficient specified in Eq.(4.9) on Page 49.
6. Update the parameters $\sigma(t)$ and $\eta(t)$ according to a pre-setted decreasing scheme and, if the learning iterations are not terminated, go to Step 3 with $t := t + 1$.

7. Calculate the activity value of each city $x_k$ according to the formula:

$$a(x_k) = m_k - \frac{3}{26} \left\{ 1 \cdot \|x_k - w_{mk}\| 
+ \frac{2}{3} \cdot [\|x_k - w_{mk+1}\| - \|x_k - w_{mk-1}\|] 
+ \frac{1}{2} \cdot [\|x_k - w_{mk+2}\| - \|x_k - w_{mk-2}\|] \right\}$$

where $m_k$ is the winning neuron corresponding to $x_k$.

8. Sort the cities by their activity values, and then form a solution of the TSP.

The first six steps comprise a special example of the ESOM algorithm in Section 4.2.2. Steps 7 and 8 constitute a realization of the solution mapping procedure of the SOM to the TSP. It aims to yield a tour from the topological ordered neurons after the SOM’s learning. Similar with the Budinich’s SOM, it can avoid the confusion problems in the solution mapping procedure [14]. Eq.(A.3) provides a mapping method which realizes how each city is mapped to a real number rather than an integer. This is designed not only for exploiting the information of the winning neuron but also benefiting from that of its nearest neurons. According to this mapping method, if there are more than one city exciting a same neuron, then the city nearer to the preceding neurons in the ring and farther from the subsequent neurons will be visited earlier in the tour. Intuitively, it could make the length of the resultant tour as short as possible. These coefficients 1, $\frac{2}{3}$, and $\frac{1}{2}$ in Eq.(A.3) are set after preliminary experiments so as to reflect the different importance of the corresponding items. The coefficient $\frac{3}{26}$ aims to limit the second item in the right hand side within $[-0.5, 0.5]$, so this item just affects the local orders of the cities exciting the same winner $m_k$.

The learning rule defined by Eq.(A.1) is the key point of the ESOM algorithm. The expanding coefficient $c_j(t)$ is specified in specified in Eq.(4.9) on Page 49, we
may use Theorem 4.3 on Page 50 to explain how well the convex hull property is reflected. First,

\[ 1 \leq c_j(t) \leq \frac{1}{\sqrt{1 - R^2}} \]

So, the algorithm is feasible. In addition, this theorem (as well as our remarks on Page 53) indicates that, roughly speaking, the expanding coefficient \( c_j(t) \) increases with \( ||x_k(t)|| \). Note that the \( ||x_k(t)|| \) is the distance of the input city from the center of all cities. In other words, the value reaches an extremum if the input city is located on the convex hull. In this case, the expanding coefficient reaches an extremum too. Then the excited neurons are greatly pushed towards the convex hull. This helps our ESOM to capture the convex hull property. On the contrary, for the cities close to the center of all cities, the expanding coefficient is small. Then the neurons corresponding to these cities will mainly follow the SOM’s learning rule to capture the local optimality.

The ESOM distinguishes itself from other previous SOMs for the TSP by its expanding learning rule. The CEN, starting from the exact convex hull, trains its neurons with the learning rule specified by Eq.(4.17) on Page 59, which emphasizes the elastic force [3]. The SOM developed by Budinich [14] employs the commonly-used learning rule as in Eq.(4.2) on Page 44. The KNIES employs the learning rule as in Eq.(4.17). It has an additional dispersing module to adjust all neurons so as to maintain some statistics explicitly. It can generate very promising solutions [5]. We will compare their performances to demonstrate the superiority of our ESOM.

### A.3 Experimental Results

To demonstrate the effectiveness and efficiency of our ESOM, a series of experiments have been conducted on both synthetic and benchmark TSPs. Some experimental results and comparison results are presented in this section.
A.3.1 Implementation

We have implemented the ESOM with C++ and executed it on a Sun Ultra 1/170 workstation. The source code and all the TSPs involved are accessible at http://www.cse.cuhk.edu.hk/~hdjin/som. The parameters used in our implementation of the ESOM are as follows:

- **Radius $R$**: is set to 0.6 according to some preliminary experiments.
- **The number of output neurons $M$**: is fixed and set equal to the number of cities $n$.
- **Training loops**: is set to 100, i.e., there are $100 \times n$ iterations.
- **The learning parameters $\eta$**: is initially set to 0.8 and is decreased linearly for each learning loop until it reaches zero at the last loop as [14].
- **The width $\sigma(t)$**: is set to $6.2 + 0.037n$ at the start and is decreased linearly to 1 in the first 65% of the iterations. For the subsequent 35% of the iterations, $\sigma(t)$ remains 1.

To evaluate the performance of the ESOM, we have compared our algorithm with 5 typical algorithms. They include the SOM proposed by Budinich [14], the enhanced Convex Elastic Net (CEN) [3], the KNIES [5], as well as the SA approach [52]. In our implementation of the Budinich’s SOM, all learning parameters have been set according to the above list. Like the implementation of the SA approach in [14], we have taken the annealing factor 0.95, adopted the exchange terms inspired by Lin and Kernigham [61], and allowed total $20 \times n$ trials at each temperature level. Most of our experiment results have been based on 10 independent runs. For convenience, we have quoted the experiment results of the enhanced CEN [3] and the KNIES [5] from papers directly rather than implementing them ourselves. It is partially due to the lack of the implementation details.
Figure A.3: Experimental results on a set of 18 synthetic TSPs: (a) Comparison of the solution qualities yielded by the ESOM, the Budinich’s SOM and the SA approach; (b) The average execution time comparison between the Budinich’s SOM and the ESOM.

A.3.2 Experimental Results

The first set of experiments is to compare the proposed ESOM with the Budinich’s SOM and the SA approach. The tested TSPs consists of 18 synthetic TSPs ranging from 50 to 2400 cities which are generated randomly within the unit square.

Figure A.3(a) shows the experimental results of the ESOM, the Budinich’s SOM, and the SA approach. The solution quality is in terms of the relative difference of the average tour length over a theoretical lower bound. The theoretical lower bound for a random TSP with $n$ cities within the unit square is calculated
by the Stein’s formula [37], i.e., $0.765 \times \sqrt{n}$. It is seen from Figure A.3(a) that the performances of the SA approach and the Budinich’s SOM are very similar, which coincides with [14]. For all 18 TSPs, the solutions obtained by the ESOM are far nearer to the theoretical bounds than those by the Budinich’s SOM and the SA approach. Its solution qualities are quite stable no matter how large TSPs are. For example, its solutions for the TSP with 2,400 cities is about 3.50% longer than the theoretical bound. Those of the SA approach and the Budinich’s SOM are 5.22% and 5.99% respectively. Specifically, we average all the results for the 18 TSPs, and find that the solution qualities are 3.93%, 6.65%, and 6.83%, respectively, for the ESOM, the Budinich’s SOM and the SA approach. Consequently, the ESOM makes around 2.72 percentage improvement on the Budinich’s SOM and 2.90 on the SA approach. Similar situations can also be observed from two other sets of experiments, as listed in Tables A.1 and A.2. There the solution quality is defined as the relative difference of the length of the generated solution from the optimal tour length. The ESOM makes at least 2.70 and 1.62 percentage improvement on its counterparts for the two sets of TSPs, respectively. Based on these series of experiments, we conclude that the ESOM has made remarkable improvement on the Budinich’s SOM and the SA approach.

Now we compare these algorithms in another measure. Figure A.3(b) shows the execution times of the ESOM and the Budinich’s SOM for the 18 TSPs. It is clear that their execution times are not significantly different, though the ESOM takes slightly longer than the Budinich’s SOM. It is noted, furthermore, that both the ESOM and the Budinich’s SOM are much faster than the SA approach. For example, it took about 15 hours to generate a solution of the TSP with 2400 cities. However, only around 10 minutes needed by the ESOM and the Budinich’s SOMs.

Our second set of experiments are primarily designed to compare the ESOM with the CEN [3], as well as the Budinich’s SOM and their enhanced versions. Here, an enhanced version of a SOM means that the neural network is improved
Table A.1: Experimental results of ESOM, the Budinich’s SOM, SA, and the enhanced CEN when applied 5 benchmark TSPs. An enhanced version is that a network is improved by the NII heuristic.

<table>
<thead>
<tr>
<th>TSP name</th>
<th>City No.</th>
<th>Optimum</th>
<th>Solution quality of 3 algorithms (%)</th>
<th>Solution quality of enhanced SOMs (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SA</td>
<td>Budinich</td>
</tr>
<tr>
<td>Gr96</td>
<td>96</td>
<td>51231</td>
<td>4.12</td>
<td>2.09</td>
</tr>
<tr>
<td>Grid100</td>
<td>100</td>
<td>100</td>
<td>2.07</td>
<td>2.17</td>
</tr>
<tr>
<td>Kroa100</td>
<td>100</td>
<td>21282</td>
<td>5.94</td>
<td>3.68</td>
</tr>
<tr>
<td>Gr137</td>
<td>137</td>
<td>69853</td>
<td>8.45</td>
<td>8.61</td>
</tr>
<tr>
<td>Lin318</td>
<td>318</td>
<td>44169</td>
<td>7.56</td>
<td>8.19</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td>5.63</td>
<td>4.95</td>
</tr>
</tbody>
</table>

with the heuristics NII trick in [3]. We have directly quoted the experimental results of the CEN directly from [3], where the CEN has been enhanced by the heuristics NII. Thus, the proposed ESOM has also been enhanced to make a fair comparison. The benchmark TSPs are taken from the TSPLIB [75]. They are accessible at www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/.

Table A.1 lists the experimental and comparison results of the plain and the enhanced SOMs. The solution quality is in terms of the relative difference of the best tour length obtained from the optimal tour length. The best tour is chosen from 10 independent runs or two runs for the SA approach. It is seen from Table A.1 that, for both the plain and the enhanced versions, the ESOM always yields much higher quality solutions of TSPs than the Budinich’s SOM, the SA approach, and the enhanced CEN(The optimal tour of GR96 cited by [3] is as long as 55209.) do. More specifically, we can observe from Table A.1 that the enhanced ESOM has improved the solution quality of the enhanced CEN from 2.95% to 1.50% on average. Even the ESOM can averagely generate better solutions than the enhanced CEN. Note that the CEN takes at least $O(n \ln n)$ additional computation to construct the convex hull explicitly, and hence the CEN would be slower than the ESOM. Therefore, we conclude that the proposed ESOM outperforms the CEN in terms of both time and accuracy.
Figure A.4: Typical solutions obtained for the TSP Att532 with 532 cities: (a) by the SA approach; (b) by the Budinich’s SOM; (c) by the ESOM.
The third set of experiments have been conducted for the comparison between the ESOM and the KNIES, which was claimed to be the most accurate neural network for the TSP currently reported in the literature [5]. Its basic idea is to, after the SOM’s learning, disperse the all neurons to make their statistics equal to that of cities. If all cities participate, it leads to a global version, denoted by KNIES-global in Table A.2. If only the represented cities are involved, it leads to a local version which is denoted by KNIES-local. Both of them perform quite well and all their results in [5] are listed in Table A.2. Here, the solution quality data again are the relative differences of the best tour lengths from the optima. All 15 TSPs can be got from the aforementioned TSPLIB. It is seen from Table A.2 that all five algorithms can generate good solutions. The solution qualities of the SA approach, the Budinich’s SOM, the KNIES-global and the KNIES-local are very close. On average, their solution qualities are 4.54%, 4.45%, 4.43% and 4.20%, respectively. The ESOM may generate higher quality solutions than its counterparts except for St70, Pr107 and Pr124. Its average solution quality is 2.81%, which make at least 1.39 percentage improvement on its counterparts. The solution qualities of the ESOM are quite stable. They range from 0.67% to 7.13%. The solutions for Att532 generated by the SA approach, the Budinich’s SOM and the ESOM are shown in Figure A.4. It is worth pointing out that the ESOM can actually achieve the convex hull property. On the other hand, both the KNIES-global and the KNIES-local need additional memory to indicate the status of each neurons. Furthermore, in order to maintain the overall statistics explicitly, they have to update each neuron in each iteration. So, the KNIES is at least more complicated than the ESOM. Consequently, the performance of the ESOM is obviously better than the two KNIES algorithms.

In summary, the ESOM outperforms the SA approach and the Budinich’s SOM both on synthetic and benchmark TSPs. With similar computational complexity, the ESOM can generate better solutions for a wide spectrum of benchmarks than the CEN and the KNIES algorithms.
Table A.2: Experimental results of ESOM, SA, the Budinich’s SOM, the KNIES-local and the KNIES-global on 15 benchmark TSPs.

<table>
<thead>
<tr>
<th>TSP name</th>
<th>City No.</th>
<th>Optimum</th>
<th>Solution quality of 5 algorithms (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eil51</td>
<td>51</td>
<td>426</td>
<td>SA: 2.33, Budinich: 3.10, KNIES-global: 2.86, KNIES-local: 2.86, ESOM: 1.10</td>
</tr>
<tr>
<td>St70</td>
<td>70</td>
<td>675</td>
<td>SA: 2.14, Budinich: 1.70, KNIES-global: 2.33, KNIES-local: 1.51, ESOM: 2.09</td>
</tr>
<tr>
<td>Eil76</td>
<td>76</td>
<td>538</td>
<td>SA: 5.54, Budinich: 5.32, KNIES-global: 5.48, KNIES-local: 4.98, ESOM: 3.89</td>
</tr>
<tr>
<td>Rd100</td>
<td>100</td>
<td>7910</td>
<td>SA: 3.26, Budinich: 3.16, KNIES-global: 2.62, KNIES-local: 2.09, ESOM: 1.96</td>
</tr>
<tr>
<td>Eil101</td>
<td>101</td>
<td>629</td>
<td>SA: 5.74, Budinich: 5.24, KNIES-global: 5.63, KNIES-local: 4.66, ESOM: 3.43</td>
</tr>
<tr>
<td>Lin105</td>
<td>105</td>
<td>14379</td>
<td>SA: 1.87, Budinich: 1.71, KNIES-global: 1.29, KNIES-local: 1.98, ESOM: 0.25</td>
</tr>
<tr>
<td>Pr107</td>
<td>107</td>
<td>44303</td>
<td>SA: 1.54, Budinich: 1.32, KNIES-global: 0.42, KNIES-local: 0.73, ESOM: 1.48</td>
</tr>
<tr>
<td>Pr124</td>
<td>124</td>
<td>59030</td>
<td>SA: 1.26, Budinich: 1.62, KNIES-global: 0.49, KNIES-local: 0.08, ESOM: 0.67</td>
</tr>
<tr>
<td>Bier127</td>
<td>127</td>
<td>118282</td>
<td>SA: 3.52, Budinich: 3.61, KNIES-global: 3.08, KNIES-local: 2.76, ESOM: 1.70</td>
</tr>
<tr>
<td>Pr152</td>
<td>152</td>
<td>73682</td>
<td>SA: 2.64, Budinich: 2.04, KNIES-global: 1.29, KNIES-local: 0.97, ESOM: 0.89</td>
</tr>
<tr>
<td>Kroa200</td>
<td>200</td>
<td>28568</td>
<td>SA: 5.61, Budinich: 6.13, KNIES-global: 6.57, KNIES-local: 5.72, ESOM: 2.91</td>
</tr>
<tr>
<td>Att532</td>
<td>532</td>
<td>27686</td>
<td>SA: 5.38, Budinich: 5.67, KNIES-global: 6.80, KNIES-local: 6.74, ESOM: 3.34</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td></td>
<td><strong>SA</strong>: 4.54, <strong>Budinich</strong>: 4.45, <strong>KNIES-global</strong>: 4.43, <strong>KNIES-local</strong>: 4.20, <strong>ESOM</strong>: 2.81</td>
</tr>
</tbody>
</table>

A.4 Summary

In this appendix, we have successfully applied the ESOM to handle the TSP. The ESOM combines successfully the topological order-preserving property of the SOM with the global optimality of the TSP — the convex hull property of the optimal tours. Through drawing the excited neuron closely towards the input city and pushing outwards from the center of all cities simultaneously in the learning procedure, the ESOM can acquire the global optimality gradually as well as construct a good topological order-preserving map. A comprehensive series of experiments for both synthetic and benchmark TSPs have shown the effectiveness and high efficiency of the ESOM. They have particularly demonstrated that the proposed ESOM outperforms the well-known simulated annealing approach and the typical SOMs such as the SOM developed by Budinich [14], the enhanced convex elastic net [3], the KNIES algorithms [5]. We believe the ESOM is one of the most accurate neural networks for the TSP in the literature.
Appendix B

ISOM for TSP

In Section 4.2.3, we have combined three learning rules together and proposed an Integrated Self-Organization Map (ISOM) for data summarization and projection. In this appendix, we give its very successful application to the Traveling Salesman Problem (TSP). We use a Genetic Algorithm (GA) to evolve an efficient ISOM automatically. The evolved ISOM (eISOM) network has been examined on a wide spectrum of TSPs. Compared with the simulated annealing approach, it can generate tours about 3% shorter. With the same quadratic computational complexity, it has made at least 1% improvement over the SOM developed by Budinich [14], our ESOM in Appendix A, the convex elastic net [3] and the FLEXMAP algorithm [32]. To the best of our knowledge, the eISOM is the most accurate SOMs for the TSP.

We present the ISOM for the TSP and some possible realizations of its learning rule in the next section. We discuss the evolutionary design of an efficient ISOM in Section B.2. The implementation of a neural-evolutionary system that evolves a promising ISOM is given in Section B.3, followed by the evolved ISOM and its performance on three sets of TSPs. We conclude the appendix in the last section.
B.1 The ISOM for the TSP

There are three promising learning rules in the SOM-like neural networks for the TSP. Their underlying ideas emphasize different aspects of the TSP. The most common learning rule given in Eq. (4.2) enables the generated tour to have the local optimality. From each city, the generated tour tries to visit its nearest neighbor as far as possible. The expanding learning rule given in Eq. (4.7) helps the tour to capture the global optimality — the convex hull property. That is, the cities located on the convex hull would be visited in the same sequence as they appear on the convex hull. The learning rule of the elastic net given in Eq. (4.17) can avoid the intersection in tours during the learning.

We can integrate these ideas together and develop a novel Integrated SOM (ISOM) network to take advantage of these three learning mechanisms. For example, the new ISOM can employ the expanding mechanism to achieve the convex hull property and explore the possible efficient interaction between the input city and the excited neuron. It can also use the elastic force constraint to inhibit intersections during learning. We present the ISOM for the TSP below and give some realizations later.

Algorithm B.1 (The ISOM for the TSP)

1. Transform the coordinates \([x'_{1i}, x'_{2i}]^T\) \((i = 1, \ldots, n)\) of all cities such that they lie within a circle centered at the origin with radius \(R(\leq 1)\). Here \(n\) is the number of the cities. Hereafter, \([x_{1i}, x_{2i}]^T\) denotes the new coordinate of \(x_i\).

2. Set \(t = 0, p = 2\), and the initial weight vectors \(w_j(0)\) \((j = 1, \ldots, n, \text{ thus } M = n)\) with random values within the circle above.

3. Select a city at random, say \(x_k(t) = [x_{1k}(t), x_{2k}(t)]^T\), and feed it to the input neurons.
4. Find the winning output neuron, say $m(t)$, nearest to $\mathbf{x}_k(t)$ according to the Euclidean metric:

$$m(t) = \arg\min_j d(\mathbf{x}_k(t), \mathbf{w}_j(t)) = \arg\min_j \|\mathbf{x}_k(t) - \mathbf{w}_j(t)\|^2. \quad (B.1)$$

5. Train neuron $m(t)$ and its neighbors within the effective width $\sigma(t)$ by using the following formula:

$$\mathbf{w}_j(t + 1) = c_j(t) \times \left\{ \mathbf{w}_j(t) + \alpha_j(t) [\mathbf{x}_k(t) - \mathbf{w}_j(t)] \right\} + \frac{\beta_j(t)}{2} [\mathbf{w}_{j-1}(t) + \mathbf{w}_{j+1}(t) - 2\mathbf{w}_j(t)] \quad (B.2)$$

$$+ \frac{\beta_j(t)}{2} [\mathbf{w}_{j-1}(t) + \mathbf{w}_{j+1}(t) - 2\mathbf{w}_j(t)] \quad (B.3)$$

where $j = m(t), m(t) \pm 1, \ldots, m(t) \pm \sigma(t)$. $c_j(t)$ is the expanding coefficient which will be discussed later. The learning rates $\alpha_j(t)$ and $\beta_j(t)$ are specified by

$$\alpha_j(t) = \eta_1(t) \times h_{j,m(t)}, \quad (B.4)$$

$$\beta_j(t) = \eta_2(t) \times h_{j,m(t)}, \quad (B.5)$$

$$h_{j,m(t)} = \begin{cases} 
1 - \frac{d_{j,m(t)}}{\sigma(t)+1} & d_{j,m(t)} \leq \sigma(t), \\
0 & \text{otherwise}. 
\end{cases} \quad (B.6)$$

Here $\eta_1(t)$ and $\eta_2(t)$ are the learning parameters of the network, $h_{j,m(t)}$ is a neighborhood function, and $d_{j,m(t)} = \text{MOD}(|j - m(t)|, n)$ is the distance between the neurons $m(t)$ and $j$ on the ring.

6. Update the effective width $\sigma(t)$, and the learning parameters $\eta_1(t)$ and $\eta_2(t)$ with predetermined decreasing schemes. If a predetermined number of loops have not been executed, go to Step 3 with $t := t + 1$.

7. Calculate the activity value of each city $\mathbf{x}_k$ according to

$$a(\mathbf{x}_k) = m_k - \frac{3}{26} \left( \|\mathbf{x}_k - \mathbf{w}_{m_k}\| + \frac{\|\mathbf{x}_k - \mathbf{w}_{m_{k+1}}\| - \|\mathbf{x}_k - \mathbf{w}_{m_{k-1}}\|}{3} \right), \quad (B.7)$$

where $m_k$ is the winning neuron associated with $\mathbf{x}_k$. 
Appendix B ISOM for TSP

8. Order the cities by their activity values, and then form a tour of the TSP.

Steps 7 and 8, similar with those in Algorithm A.1, generates a tour from a trained SOM. Step 1 is a linear transformation. It moves the center of all cities to the origin, and restricts all cities within a circle with radius $R$. This transformation does not influence the solution space of the TSP. This step mainly facilitates the implementation of the expanding coefficient $c_j(t)$ and makes it possible to reflect the convex hull based only on the input city and the excited neuron. After this linear transformation, the distance between the city and the origin, namely the norm of the input city, is proportional to the distance between the original city and the center of all cities. Thus, the norm of the input city can be used to reflect the location of the city with respect to all cities. If the norm is larger, the city is more likely to locate on the convex hull. We then can formulate the expanding coefficient $c_j(t)$ to reflect the convex hull property in such a way that $c_j(t)$ increases with the norms of input city and the excited neuron. Furthermore, since the input city and the excited neuron are within the unit circle and they are close to each other, their inner product is close to their norms. That means, the inner product may also be used to reflect the relative locations of the input city and the excited neuron. Thus, using the norms and the inner products, we can design some reasonable expanding coefficients to reflect the convex hull.

The learning rule in Eq.(B.2) is the key point of the proposed ISOM. It also distinguishes the ISOM from all previous SOMs. The learning rule is illustrated in Figure 4.11(b). First, the excited neuron is dragged towards the input city. This adaptation is indicated by the expression enclosed in curved brackets. The expression is the same as that at the right-hand side of Eq.(4.2). In Figure 4.11(b), the neuron reaches $w'_j(t + 1)$, which behaves like the adaptation in Figure 4.6(a). Secondly, the excited neuron is pushed away from the origin, as specified by the expanding coefficient $c_j(t)(\geq 1)$. As shown in Figure 4.11(b), the neuron moves
from $\mathbf{w}_j(t + 1)$ to $\mathbf{w}_j(t + 1)$. This adaptation, similar to that in Figure 4.6(b), may be viewed as pushing the neuron to the convex hull of the TSP since the convex hull surrounds the origin. Thus, it helps the ISOM to make tours visit the cities on the convex hull in the same sequence as these cities appear on the convex hull. Finally, the excited neuron is drawn by the elastic force as indicated by the last item in Eq. (B.2). In Figure 4.11(b), the neuron moves from $\mathbf{w}_j''(t + 1)$ to $\mathbf{w}_j(t + 1)$. This adaptation is similar to the one in Figure 4.11(a). It is clear that the rule embodies the three learning mechanisms.

The expanding coefficient

We turn to formulate the expanding coefficient $c_j(t)$ in detail. To clarify the procedure, we express the expanding coefficient $c_j(t)$ with several parts that have respective functionalities. It is formulated as

$$c_j(t) = [1.0 + b_j(t) \times e_j(t)]^{a_4}.$$  \hspace{1cm} (B.8)

The constant 1.0 ensures that the expanding coefficient is close to 1.0 so as to make our learning rule harmonize well with the one in Eq. (4.2). The constant $a_4$ unifies the expanding coefficient $c_j(t)$ with the one of the ESOM in Eq. (4.7). The item $b_j(t)$ is used to adjust the relative strength of the expanding force with respect to the learning rate $\alpha_j(t)$. In other words, it harmonizes the expanding force and the dragging force. To unify the expanding coefficient $c_j(t)$ in Eq. (B.8) with the one of the ESOM network in Eq. (4.7), $b_j(t)$ is formulated as

$$b_j(t) = a_1 \times \alpha_j(t)^{a_2} \times (1 - \alpha_j(t))^{a_3}$$  \hspace{1cm} (B.9)

where parameters $a_i$ ($i = 1, 2, 3$) are positive numbers. The item $e_j(t)$ in Eq. (B.8) reflects the convex hull in terms of the vector properties of the input city and the excited neuron. As analyzed above, the norm of a city (or neuron) in the ISOM can reflect the location of the city (or neuron) with respect to other cities. The larger its norm is, the more likely the city (or neuron) locates on the convex hull. On the other hand, since both the city and the neuron are restricted within the
circle with radius $R$, their inner product will approach their norms. Thus, the inner product can be used in the expanding coefficient to differentiate the roles of different cities, too. Together with the neighborhood preserving property, this expansion helps the ISOM to generate tours which visit the cities on the convex hull in the same sequence as the cities appear on the convex hull. In other words, the ISOM can reach the global optimality. Consequently, the ISOM may generate better tours. The norm of city (or neuron) and the inner product can be used to form a lot of implementations of the item $e_j(t)$. We list some implementations used in our experiments below.

1. $e_j(t) = \|w'_j(t+1)\|^2 - |\langle x_k(t), w_j(t) \rangle|$, is the difference between the distance of the temporary neuron $w'_j(t+1)$ from the origin and the absolute value of the inner product of the input city $x_k(t)$ and the excited neuron $w_j(t)$. 

2. $e_j(t) = \|w'_j(t+1)\|^2 + \|x_k(t) - w_j(t)\|^2$, is the sum of the distance of the temporary neuron $w'_j(t+1)$ from the origin and the distance of the city $x_k(t)$ from the neuron $w_j(t)$. 

3. $e_j(t) = \|x_k(t) - w_j(t)\|^2 \times \|x_k(t)\|^2$, is the product of the distance of the city $x_k(t)$ from the origin and the distance of the city $x_k(t)$ from the neuron $w_j(t)$. 

4. $e_j(t) = \|w_j(t)\|^2 - \langle x_k(t), w_j(t) \rangle$, is the difference between the distance of the neuron $w_j(t)$ from the origin and the inner product of the city $x_k(t)$ and the neuron $w_j(t)$. 

5. $e_j(t) = \|x_k(t)\|^2 - \langle w_j(t), x_k(t) \rangle$, is the difference between the distance of the city $x_k(t)$ from the origin and the inner product of the city $x_k(t)$ and the neuron $w_j(t)$.

Since the proposed ISOM integrates three learning mechanisms together, an efficient ISOM must have good coordination among the local optimality of the
traditional SOM, the global optimality of the expanding coefficient, and the elastic force constraint. Moreover, a suitable implementation of the expanding coefficient and parameter settings should be determined. It seems very difficult to specify a good ISOM manually. Thus, we employ a genetic algorithm in this appendix to design an efficient ISOM.

### B.2 Evolutionary Design of the ISOM

During the past two decades there has been growing interest in evolutionary algorithms, especially Genetic Algorithm (GA). They are a family of global stochastic search algorithms based on Darwin’s theory of evolution (survival of the fittest) [36, 89]. Normally, these algorithms maintain a population of chromosomes, and manipulate them by using several genetic operators. A schematic view of a GA is given in the right-hand side of Figure B.1. The most significant advantages of using evolutionary algorithms lie in the gain of flexibility and adaptability to
the task on hand, in combination with robust performance and global search characteristics. Thus they have been employed to handle many inherently hard problems. As mentioned in [81, 89, 96], the search space of all possible network structures (size and connectivity) and learning rules is infinitely large, un-differentiable, deceptive and multi-modal. Evolutionary algorithms provide promising and automatic alternatives to solve such a difficult problem as designing neural networks. Moreover, the combination of the learning and the evolutionary mechanisms can significantly improve the trainability, the productivity, and the problem-solving capability of the neural-evolutionary systems.

In a neural-evolutionary system, evolution and learning are the two fundamental adaptation mechanisms. Evolution can be introduced into neural networks on various levels. It can be used to evolve the termination criteria [81], weights [96], architecture [19, 70], and learning rules [87]. It is hard to say which one is on a higher level [96].

Since the architecture of the SOMs for the TSP has been well-studied, we concentrate on evolving a learning scheme that consists of a learning rule, the parameter setting of the learning rule, and the learning parameter setting. Once the architecture of a network is known, a learning scheme can generate a concrete neural network algorithm. The algorithm can be used to handle several TSPs and the evolutionary algorithm will not be executed again. Our approach is different from the one that evolves the architecture and learns the synaptic weights together. In the latter approach, both evolution and learning are employed in each target problem.

For the problem of evolving a good learning scheme, the performance of a candidate learning scheme acts as its fitness. The performance is estimated by considering the speed, the accuracy, and the generalization capability of the learning scheme. In order to obtain an accurate estimation of the fitness value, a large number of different target problems should be solved by using the scheme. Obviously, this fitness evaluation procedure will take a long time to evaluate a
scheme. Furthermore, the fitness evaluation procedure must be executed for different schemes during the evolution process. Thus, it will take extremely long to evolve a learning scheme if an accurate fitness evaluation procedure is used.

In order to handle this problem, the fitness value of a learning scheme may be estimated by applying the learning scheme to one or a few small-scale target problems. However, this approach will introduce the noisy fitness evaluation problem because the fitness value of a scheme relies on the selected target problems. To alleviate the noisy fitness evaluation problem, a learning scheme is examined on the target problems for several runs and our fitness function considers the average performance and the variance of performance among these runs.

Since we cannot ensure that the learning scheme with the best fitness value on a few small-scale target problems also performs well on all target problems, we introduce a validation set of target problems to verify the generalization capability of the learning scheme. It is expected that a learning scheme with good performance on the validation set of problems will also perform well on other problems.

A neural-evolutionary system that evolves a learning scheme is shown in Figure B.1. After initialization, a genetic algorithm is used to evolve good learning schemes. A concrete neural network algorithm is obtained from a learning scheme. The neural network algorithm is then used to solve a number of small-scale target problems in order to estimate the fitness value of the corresponding learning scheme. The best $Q$ different schemes during the evolution process are stored. The evolution process iterates until the termination criterion is satisfied and the stored learning schemes are verified on a set of large-scale problems. Finally, the learning scheme with the best fitness on the validation problems is returned as an evolved ISOM (eISOM).
Table B.1: 13 alleles of an individual and their domains in the neural-evolutionary system, and the parameter setting in the evolved ISOM (eISOM). The learning parameters $\eta_1(t)$, $\eta_2(t)$ and $\sigma(t)$ are decreased linearly after each learning iteration. $\eta_1(t)$ reaches zero at the last loop.

<table>
<thead>
<tr>
<th>Alleles in an individual</th>
<th>Domains</th>
<th>eISOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formula for $e_j(t)$</td>
<td>${1, 2, \cdots, 20}$</td>
<td>1</td>
</tr>
<tr>
<td>Parameters for $e_j(t)$:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1$, $a_2$ and $a_3$</td>
<td>${0.0.25, \cdots, 5}$</td>
<td>1,3,0.25</td>
</tr>
<tr>
<td>$a_4$</td>
<td>${0.2, 0.4, \cdots, 5}$</td>
<td>1.0</td>
</tr>
<tr>
<td>Radius $R$</td>
<td>$(0.001, 1.0)$</td>
<td>0.61</td>
</tr>
<tr>
<td>Learning loop $L$</td>
<td>${50, 65, \cdots, 200}$</td>
<td>160</td>
</tr>
<tr>
<td>Learning parameter $\eta_1(0)$</td>
<td>$(0.001, 1.0)$</td>
<td>0.95</td>
</tr>
<tr>
<td>Learning parameter $\eta_2(0)$</td>
<td>$(0.001, 1.0)$</td>
<td>0.12</td>
</tr>
<tr>
<td>$\eta_2(t)$ decreasing mode (be 0 after $p_1$ percent iterations)</td>
<td>$p_1$: (0,100)</td>
<td>48</td>
</tr>
<tr>
<td>The effective width</td>
<td>$a$: ${1, 2, \cdots, 14}$</td>
<td>10</td>
</tr>
<tr>
<td>$\sigma(0) = a + b \cdot n$</td>
<td>$b$: $[0.001, 0.6]$</td>
<td>0.01</td>
</tr>
<tr>
<td>$\sigma(t)$ decreasing mode (be 1 after $p_2$ percent iterations)</td>
<td>$p_2$: (0,100)</td>
<td>62</td>
</tr>
</tbody>
</table>

B.3 Implementation and Experimental Results

We have implemented the above neural-evolutionary system to evolve an efficient ISOM for the TSP. All algorithms are implemented in C++ and all experiments are performed on a Sun Ultrasparc 5/270 workstation.

B.3.1 Evolving the ISOM

We have used the canonical GA in the neural-evolutionary system. In the GA, every individual represents a learning scheme. Its 13 alleles code 13 parameters. The parameters include the type of formula to calculate the expanding coefficient and the parameters $a_i$ ($i = 1, \cdots, 4$). They also include other parameters in the ISOM such as the radius $R$, the total learning loop $L$, and the initial values and the decreasing schemes of the effective learning width $\sigma(t)$ and the learning
parameters $\eta_1(t)$ and $\eta_2(t)$. Thus, given the ISOM discussed in Section B.1, an individual determines a concrete ISOM. These 13 alleles and their domains are listed in Table B.1. Our GA ensures that all alleles stay in their respective domains, thus invalid individuals will never be generated.

We use the relative difference between the generated tour length and the optimal tour length to measure the solution quality. The fitness function is designed to indicate the average solution quality as well as its consistency on several runs. It is formulated as

$$fitness = 3 - \text{ave}(\text{quality}) - \text{var}(\text{quality}).$$

(B.10)

Here $\text{ave}(\text{quality})$ and $\text{var}(\text{quality})$ are the average solution quality and its variance respectively. If the optimal tour length is unknown, we use a theoretical lower bound in place of the optimal tour length. The theoretical lower bound says that the shortest tour length for a random TSP with $n$ cities within the unit square is close to $0.765 \sqrt{n}$ [37].

The roulette wheel selection method is used to select parents in each generation [36]. The mutation operator modifies the old allele value to a random value in the domain. Two crossover operators are applied alternatively. The first one is the one-point crossover operator that exchanges the alleles after a random point in two parents. The second one considers each allele in turn and generates a random value that is close to the two corresponding allele values in the parents. For example, if the allele values of the parents are $z_1$ and $z_2$ respectively, the allele value $z_o$ of the offspring will be $z_2 + \lambda(z_2 - z_1)$, where $\lambda$ is a random value in $[-1.0, 1.0]$. If the value $z_o$ is out of the domain, it will be changed to the closest valid value.

In our neural-evolutionary system, the crossover and the mutation probabilities are 0.99 and 0.01, respectively. The population size is 100 and the maximum number of generations is 6000. The fitness value is evaluated on three runs of two random TSPs with 30 and 50 cities, respectively. During
the evolution process, the best 30 individuals are stored. The stored individuals are then evaluated on three runs of three random TSPs with 200, 800, 1800 cities, respectively. All random TSPs used in this appendix can be found at http://www.cse.cuhk.edu.hk/~hdjin/som/.

The learning scheme evolved is listed in the last column of Table B.1. The executable program and the source codes of the evolved ISOM (eISOM) network can also be downloaded at the above web site. The explanation of the evolved learning scheme is given as follows.

- **The expanding coefficient** $c_j(t)$ is

  \[
  c_j(t) = 1 + \alpha_j(t)^3 (1 - \alpha_j(t))^{0.25} \left\{ \sum_{i=1}^{2} [\alpha_j(t)x_{ik}(t) + (1 - \alpha_j(t))w_{ij}(t)]^2 - \left| \sum_{i=1}^{2} x_{ik}(t)w_{ij}(t) \right| \right\}.
  \]  
  \[(B.11)\]

  That means, $e_j(t)$ in Eq. (B.8) is calculated using the first formula listed.

- **Radius** $R$ is 0.61.

- **Learning loop** $L$ is set to be 160. Thus, each city is circularly fed into the ISOM 160 times. Namely, there are totally $160 \times n$ learning iterations.

- **The learning parameter** $\eta_1(t)$ is initialized to 0.95 and is decreased linearly for each learning loop until it reaches zero at the last loop.

- **The learning parameter** $\eta_2(t)$ is initialized to 0.12 and is decreased linearly to 0 in the first 48% learning iterations.

- **The effective width** $\sigma(t)$ is initialized to $10 + 0.01n$ and is decreased linearly to 1 in the first 62% of the iterations. It keeps 1 in the remaining 38% iterations.

The expanding coefficient $c_j(t)$ in Eq.(B.11) only consists of the learning rate $\alpha_j(t)$, the input city $x_k(t)$ and the excited neuron $w_j(t)$. The expanding
item $e_j(t) = \|w_j(t+1)\|^2 - |<w_j(t),x_k(t)>|$ is used to reflect the convex hull. Roughly, the closer the input city (or the excited neuron) to the convex hull, the larger the expanding item is. That means, the cities on the convex hull have more influence during learning. So, together with the neighborhood preserving property, the expanding coefficient can helps the ISOM to generate shorter tours. On the other hand, the expanding coefficient $c_j(t)$ equals 1.0 either when the excited neuron is identical to the input city or when the learning rate $\alpha_j(t)$ reaches 0. Thus, when the learning rate $\alpha_j(t)$ approaches zero, the expanding coefficient has no influence on learning. In other words, the evolved ISOM has similar asymptotic behavior to the SOM because the learning rates $\alpha_j(t)$ and $\beta_j(t)$ finally approach zero. Therefore, the evolved expanding coefficient is reasonable.

### B.3.2 Performance of the Evolved ISOM

The computational complexity of the evolved ISOM is $O(n^2)$, which is the same as the ESOM [59] and the SOM developed by Budinich [14]. It is worth noting that almost all non-neural network methods, such as simulated annealing (SA) [52], GAs [82] and ant colony systems [24], have a higher complexity. Thus, they take longer time to find tours comparable with those generated by the SOMs.

To evaluate the performance of the evolved ISOM, we have compared it with several typical algorithms including the SA approach [52], the SOM developed by Budinich [14], the ESOM [59], the Convex Elastic Net (CEN) [3], and the FLEXMAP algorithm [32] on three sets of TSPs. For the SA approach, the annealing factor 0.95 is used as in [14]. The exchange method, known as 2-opt inspired by Lin and Kernigham [61], is adopted. The SA approach allows $20 \times n$ trials at each temperature level. It usually generates better tours than the heuristic 2-opt [14]. The Budinich’s SOM, an effective implementation of the canonical SOM, maps each city onto a linear order without ambiguity. We set all parameters according to Budinich [14], which can generate tours comparable
Appendix B ISOM for TSP

Figure B.2: (a). Comparison of the solution quality of the evolved ISOM, the ESOM, the Budinich’s SOM and the SA approach on 18 random TSPs; (b). The average execution time comparison among the three SOMs.

to the SA approach. The ESOM network uses the learning rule in Eq.(A.1), and we implement it using the parameter setting in [59]. The CEN algorithm has to form the convex hull of the given cities explicitly. It then takes the initial tour on the convex hull and trains the network in a similar way as the elastic net. But it yields shorter tours of TSPs than the elastic net [3]. We have not implemented CEN due to the lack of its implementation details. The experiment results of CEN below are quoted from [3]. The FLEXMAP algorithm inserts a new neuron in the ring of the neurons every several learning loops. We also quote its performance results directly from the paper [32].
The first set of experiments were conducted on a set of 18 TSPs with from 50 to 2400 cities. These TSPs are all generated randomly within the unit square. Figure B.2(a) shows the experiment results of the eISOM, the ESOM, the Budinich’s SOM and the SA approach. The solution quality is represented in terms of the relative difference between the average tour length and the theoretical lower bound. The results are based on 10 runs.

From Figure B.2(a), it can be observed that the tours generated by the eISOM are much nearer to the theoretical bounds than those by both the SA approach and the Budinich’s SOM. Except the tours for the TSP with 400 cities, the tours generated by the eISOM are shorter than the ones generated by the ESOM on average. The solution quality of the eISOM varies slightly with the sizes of the TSPs. For example, its tours for the TSP with 2,400 cities is about 1.59% longer than the theoretical bound. The ESOM network generates tours 3.50% longer than the theoretical bound, the Budinich’s SOM generates tours 5.99% longer than the theoretical bound and the SA approach generates tours 5.24% longer than the theoretical bound. The eISOM performs substantially better than its three counterparts. Figure B.3 depicts the typical tours generated by these four algorithms. It is interesting to point out that the tour generated by the eISOM visits the cities on the convex hull in the same sequence as these cities appear on the convex hull. Moreover, it has no intersections as illustrated in Figure B.3(d). However, the three tours generated by its counterparts all intersect themselves.

We have averaged all the experiment results on these 18 TSPs. The average relative difference is 2.63% for the eISOM. In other words, the tours obtained are 2.63% longer than the theoretic lower bounds on average. The average relative differences are 3.93% for the ESOM, 6.65% for the Budinich’s SOM, and 6.83% for the SA approach respectively. Consequently, the eISOM makes 1.30% improvement over the ESOM, and makes around 4% improvement over the Budinich’s SOM and the SA approach.

The execution times of the three SOMs are illustrated in Figure B.2(b). The
Table B.2: Experiment results of the SA approach, the Budinich’s SOM, the ESOM, and the evolved ISOM (eISOM), and the enhanced CEN algorithm when applied to 5 TSP benchmarks.

<table>
<thead>
<tr>
<th>TSP name</th>
<th>number of Cities</th>
<th>Optimum</th>
<th>Solution quality of 4 algorithms (%)</th>
<th>Solution quality of enhanced SOMs (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GR96</td>
<td>96</td>
<td>51231</td>
<td>4.12 2.09 1.03 0.81</td>
<td>4.39 0.46 0.46 0.53</td>
</tr>
<tr>
<td>GRID100</td>
<td>100</td>
<td>100</td>
<td>2.07 2.17 0.83 0.83</td>
<td>0.80 1.63 0.80 0.80</td>
</tr>
<tr>
<td>KROA100</td>
<td>100</td>
<td>21282</td>
<td>5.94 3.68 1.01 0.57</td>
<td>1.60 0.93 0.81 0.54</td>
</tr>
<tr>
<td>GR137</td>
<td>137</td>
<td>69853</td>
<td>8.45 8.61 4.27 3.16</td>
<td>3.29 4.51 2.52 2.18</td>
</tr>
<tr>
<td>LIN318</td>
<td>318</td>
<td>44169</td>
<td>7.56 8.19 4.11 2.05</td>
<td>4.67 2.81 2.89 1.96</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>5.63</td>
<td>4.95 2.25 1.48</td>
<td>2.95 2.07 1.50 1.20</td>
</tr>
</tbody>
</table>

The execution times increase linearly with the sizes of TSPs. The execution time of the eISOM is about 1.6 times of that of the ESOM and the Budinich’s SOM for each TSP. It is mainly due to that the eISOM has more learning loops to execute. It should be emphasized that the three SOMs are much faster than the SA approach. For example, the SA approach spends about 5,400 seconds to generate a solution of the TSP with 2400 cities. However, the eISOM spends about 1,000 seconds, and the other SOMs spends about 600 seconds.

The second set of experiments were mainly designed to compare the eISOM with the Convex Elastic Net (CEN) of Al-Mulhem and Al-Maghrabi [3]. We also present the experiment results of the SA approach, the Budinich’s SOM, the ESOM and their enhanced versions. An enhanced version is that a network is improved by the NII heuristic. The NII heuristic, used in [3], improves tours by using a rearrangement heuristic derived from 2-opt. Since the experiment results of the CEN algorithm quoted from [3] have been enhanced by the NII heuristic, the other SOMs have also been enhanced to make a fair comparison. We tested the algorithms on 5 TSP benchmarks examined by the CEN algorithm. The five TSPs can be taken from the TSPLIB collected by Reinelt [75].

Table B.2 lists the experiment results of the original and enhanced SOMs. The results are based on 10 independent runs and are presented in terms of
the relative differences between the average tour lengths and the optimal tour lengths. It can be observed from Table B.2\textsuperscript{1} that the eISOM always yields better solutions than the SA approach, the Budinich’s SOM, and the ESOM. On average, the eISOM makes 0.77\% improvement over the ESOM, and makes about 3.5\% improvement over the Budinich’s SOM and the SA approach. These results accord with the first set of experiments. The enhanced eISOM obtains shorter tours than other enhanced neural networks for the problems except for GR96. On average, the enhanced eISOM generates tours 1.20\% longer than the optima. The enhanced CEN algorithm generates tours 2.95\% longer than the optima. The enhanced ESOM and the Budinich’s SOM generate tours 1.50\% and 2.07\% longer than the optima respectively. The enhanced eISOM performs better than three other algorithms. Since the computational complexity of CEN is also $O(n^2)$, we conclude that the enhanced eISOM outperforms the enhanced CEN algorithm.

The third set of experiments were performed to compare the eISOM with the FLEXMAP algorithm\textsuperscript{[32]}. The experiment results are listed in Table B.3\textsuperscript{2}. All 10 TSPs can be found in the TSPLIB. The listed results for each TSP are the relative differences between the best tour length and the corresponding optimum. They are based on 20 runs for each problem. The results of the FLEXMAP algorithm is quoted from \textsuperscript{[32]}. An enhanced version means that the algorithm is improved by the local improvement heuristic used by the FLEXMAP algorithm \textsuperscript{[32]}. This heuristic computes all 24 permutations of every subtour with 4 cities and employs the shortest permutation in order to get a better tour. The experiment results for 4 enhanced algorithms are listed in the last four columns of Table B.3.

It can be observed from Table B.3 that the enhanced eISOM generates shorter tours than the FLEXMAP algorithm, the enhanced ESOM and the Budinich’s SOM for all TSPs except for the EIL51, the EIL101 and the PCB442 problems.

\textsuperscript{1}The optimal tour of GR96 cited by [3] is as long as 55209.
\textsuperscript{2}The optimal tour length of HT30 is 415 according to [28] and our experiments. And the optimal tour of ATT532 cited by [32] is as long as 27686.
Table B.3: Experiment results of the SA approach, the FLEXMAP algorithm, the Budinich’s SOM, the ESOM, and the evolved ISOM (eISOM) when applied to the third set of TSPs.

<table>
<thead>
<tr>
<th>TSP name</th>
<th>Number of cities</th>
<th>Optimum</th>
<th>Solution quality of 4 algorithms (%)</th>
<th>Solution quality of 4 enhanced SOMs (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>SA</td>
<td>Budinich</td>
</tr>
<tr>
<td>HT30</td>
<td>30</td>
<td>415</td>
<td>0</td>
<td>1.51</td>
</tr>
<tr>
<td>EIL51</td>
<td>51</td>
<td>426</td>
<td>2.33</td>
<td>3.10</td>
</tr>
<tr>
<td>EIL101</td>
<td>101</td>
<td>629</td>
<td>5.74</td>
<td>5.24</td>
</tr>
<tr>
<td>KROA150</td>
<td>150</td>
<td>26524</td>
<td>4.31</td>
<td>4.36</td>
</tr>
<tr>
<td>KROA200</td>
<td>200</td>
<td>29368</td>
<td>5.61</td>
<td>6.13</td>
</tr>
<tr>
<td>LK318</td>
<td>318</td>
<td>42029</td>
<td>7.56</td>
<td>8.19</td>
</tr>
<tr>
<td>PCB442</td>
<td>442</td>
<td>50779</td>
<td>9.15</td>
<td>8.43</td>
</tr>
<tr>
<td>ATT532</td>
<td>532</td>
<td>87550</td>
<td>5.38</td>
<td>5.67</td>
</tr>
<tr>
<td>TK1002</td>
<td>1002</td>
<td>259045</td>
<td>7.32</td>
<td>8.75</td>
</tr>
<tr>
<td>TK2392</td>
<td>2392</td>
<td>378032</td>
<td>8.18</td>
<td>10.26</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td>5.56</td>
<td>6.16</td>
</tr>
</tbody>
</table>

The average relative difference for the enhanced eISOM is 2.72%. The average relative differences are 4.37%, 3.17% and 4.41% for the enhanced versions of the FLEXMAP algorithm, the ESOM and the Budinich’s SOM respectively. The enhanced eISOM makes 1.65% improvement over the FLEXMAP algorithm. Furthermore, the eISOM performs very well even without being enhanced by the local improvement heuristic. Observed from Table B.3, the average relative difference for the eISOM is close to that for the enhanced ESOM network. It is smaller than that for the FLEXMAP algorithm. The eISOM makes 1.13% improvement over the FLEXMAP algorithm. The improvement is promising because the tours obtained by the FLEXMAP algorithm are very near to the optima.

In summary, for these three comprehensive sets of experiments, the eISOM can generate about 3% shorter tours than the SA approach with respect to the optima. For the average relative differences, it makes at least 1% improvement over the other four SOMs on a wide spectrum of TSPs with the same computational complexity. To the best of our knowledge, it is one of the most accurate SOMs for the TSP.
B.4 Discussion

In this appendix, we have developed the Integrated Self-Organizing Map (ISOM) network, a new Self-Organizing Map (SOM) network for the TSP. Its learning rule has embodied three effective learning mechanisms of different SOMs. The rule simultaneously takes account of the local optimality of the traditional SOM, the global optimality of the Expanding SOM (ESOM), and the elastic force constraint in the elastic net. This learning rule enables the ISOM to generate near optimal solutions.

Since an efficient ISOM must have good coordination among the three learning mechanisms and use a suitable implementation of the expanding coefficient and parameter setting, it is very difficult to design a good ISOM manually. We have used a Genetic Algorithm (GA) to evolve an efficient ISOM automatically. The evolved ISOM (eISOM) network has been examined on a wide spectrum of TSPs. Compared with the simulated annealing approach, it can generate tours about 3% shorter. With the same quadratic computational complexity, it has made at least 1% improvement over the SOM developed by Budinich [14], the ESOM [59], the convex elastic net [3] and the FLEXMAP algorithm [32]. The eISOM is one of the most accurate SOMs for the TSP.

This research not only supports that GAs can be used to solve complicated problems effectively, but also substantiates the idea of the ISOM to integrate strengths of different learning mechanisms. We expect to use this methodology to handle other problems such as cluster analysis. We also expect to enhance the performance of the SOMs by embodying another global property in the learning rules.
Figure B.3: 4 typical tours of the random TSP with 2400 cities obtained by (a) the SA approach, (b) the Budinich’s SOM, (c) the ESOM, (d) the evolved ISOM.
Bibliography


[34] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison L. Powell, 
and James C. French. Clustering large datasets in arbitrary metric spaces. 
In *Proceedings of the 15th International Conference on Data Engineering*, 


[38] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE - an efficient 
clustering algorithm for large databases. In *Proceedings of ACM-SIGMOD 


[41] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan 


