A Genetic Algorithm Based Approach for Clustering Categorical Data

By

Lee Ho Kei, Sean

Master of Philosophy
in
Department of Computing
The Hong Kong Polytechnic University

Jun 2006

A thesis submitted in partial fulfilment of the requirements for the Degree of Master of Philosophy

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

_____(Name of student)

Abstract of thesis entitled ‘A Genetic Algorithm Based Approach for Clustering Categorical Data’ submitted by Lee Ho Kei, Sean for the degree of Master of Philosophy at The Hong Kong Polytechnic University in May 2006.

Abstract

Given a database of records, clustering is concerned with the grouping of similar records into different groups or clusters based on their attribute values. Many algorithms have been proposed in the past to address the clustering problem but most of them are developed mainly to handle continuous-valued data. Relatively little attention has been paid to the clustering of categorical data. Given that these kind of data is very commonly collected in many applications in business, medicine and the social sciences, etc., it is important that an effective clustering algorithm be developed to handle such data, in this thesis, we propose such an algorithm. This algorithm is based on the use of a simple genetic algorithm (GA) that employs a probabilistic search technique for solutions that are supposedly optimal or near-optimal according to some performance criteria. This GA-based clustering algorithm makes use of an encoding scheme that can encode clustering results in chromosomes effectively. To work with this scheme, we also propose a set of genetic operators that can facilitate the exchange of clustering information between chromosomes on one hand and allow variations to be introduced on the other. For the proposed GA to work well, we have also introduced a fitness function to evaluate clustering quality. This is based on an information theoretic measure that measures how much the presence of a particular attribute value supports or refutes a record in a data set to be classified into a specific cluster. The higher its fitness value based on the evaluation function, the better the solution encoded in a chromosome. Unlike traditional algorithm, the proposed GA-based clustering algorithm has the advantage that it can

automatically determine the number of clusters hidden in a dataset. The proposed algorithm has been tested with both simulated and real data; the results show that it is very promising and can have many real applications.

Acknowledgements

I would like to express my sincere thanks to all those who give me supports in finishing this thesis including my dearest friends and family. I also wish to thank Professor Keith Chan for his patience and guidance throughout the whole project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Given a set of records each characterized by a common set of attributes, clustering is concerned with the grouping together of similar records based on their attribute values. Clustering is sometimes also known as unsupervised learning, numerical taxonomy, vector quantization, and learning by observation [Jain *et al.* 1999]. Over the last several decades, many clustering algorithms have been developed and used with much success in many applications such as scientific data exploration, information retrieval, text mining, spatial database applications, computer vision, web analysis, CRM marketing, medical diagnostics, and computational biology. More recently, clustering has been used in data mining to deal with data that typically finds in database or data warehouse of large enterprises [Fayyad *et al.* 1996]. These databases or data warehouse typically consist of not just continuous-valued numerical data, but also large amount of data characterized by categorical variables.

A categorical variable is a variable that can take on two or more values, for example, *colour* can take on {red, yellow, green, …} and *weather conditions* can take on {sunny, rainy cloudy,…}, etc. Many clustering algorithms have been proposed in the past to handle numeric data and relatively little attention have been given to categorical data. Since the proximity measure used in traditional clustering algorithms cannot be defined directly on categorical data, these algorithms do not handle categorical data clustering effectively. Given that such data are so commonly

found in different application areas such as those in business, medicine and the social sciences, it is important that an effective algorithm can be developed for these data.

1.1. The Clustering Problem

For a clustering algorithm to perform effectively with categorical data, it has to tackle a number of problems including: (i) determining the number of clusters to be formed when given a data set; (ii) handling noisy and missing data values; and (iii) allowing clustering results to be interpreted and (iv) processing capability to deal with categorical data.

Determining the number of clusters to be formed

One problem that faced by many clustering algorithms is to determine how many clusters to form in a given data set. Many popular clustering techniques such as the *K*-means algorithm, requires that the desired number of clusters to be formed to be decided by the users. Providing such information is often difficult as there is usually a lack of enough domain knowledge for such decision. Because of this difficulty, some effort has been made to look into the determination of the "right" number of clusters [Cristofor and Simovici 2002; Everitt *et al.* 2001; Fraley and Raftery 1998; and Tibshirani *et al.* 2000]

Difficulties in handling missing and noisy values

Real-world data are often incomplete and inconsistent. Missing and noisy values are common in all kinds of databases [Han and Kamber 2001]. Missing values can be found for numerous reasons. For example, it is possible that relevant information may not be available at the time of collection, or may not be regarded as important at the time of entry. They may also be due to human error such as omission. Other than missing values, dirty or noisy data can be recorded as a result of inconsistency of data sources, or randomness or variations when measurement data are obtained. They may also be introduced as a result of deliberately supply of false information.

Many clustering techniques do not perform well with the presence of too many noisy values in the data. For example, for centroid based clustering algorithm such as the

K-means algorithm, if the initial centers are not chosen properly, the noisy values could cause splitting of a cluster or merging of two supposedly distinct clusters into one. This problem is complicated by the fact that many clustering algorithms do not automatically determine the number of clusters that should be formed. As a result, noisy and missing values can significantly distort the results of some clustering algorithms.

To avoid the problems caused by noisy values, a data set has to be preprocessed by data cleaning or data cleansing procedures [Fayyad *et al.* 1996a, 1996b; and Fayyad 1998a]. However, no data cleaning method can filter data perfectly and they may not be suitable for all problem domains. For instance, given a clinical database, the missing of some symptoms may actually provide useful information. In this case, the use of typical data cleaning and preprocessing techniques, such as inserting normalized data into the missing field, modifying prior probabilities, or using average data values [Kennedy *et al*. 1997], etc., may grossly distort the original data [Matthews and Hearne 1991]. For a clustering algorithm to effectively perform its tasks, there is a need for it to be able to handle noisy and missing values as much as possible.

Difficulties in interpreting clustering results

Data interpretation is very important in many data mining tasks [Fayyad *et al.* 1996a, 1996b, 1996c]. However, for clustering, the basis on which data assignments are made is not always clear. It is also not always clear what knowledge different clustering techniques discover. For example, clustering techniques based on the use of distance metrics do not interpret clustering results but merely provide grouping information. Some clustering techniques, such as hierarchical agglomerative clustering techniques [Kaufman and Rousseeuw 1990] do generate hierarchies but does not necessarily allow easy interpretation of the nature of patterns revealed inside the hierarchy. One way to deal with this interpretation problem is to use a separate inductive learning algorithm, such as C4.5 [Quinlan 1993] to generate a decision tree that can, hopefully, allow patterns underlying each cluster to be explicitly described. While this can help understand differences between clusters,

these inductive learning techniques are normally designed quite differently from the algorithm that generates the clusters. The rules that are induced when using C4.5 may not describe a cluster perfectly.

Problems with categorical data

Many clustering algorithms have mainly been developed to deal with continuous-valued data. Typically, a distance measure defined in the Euclidean space are used to determine if two data records are similar or close enough to be placed in the same clusters. However, distance measure defined in the Euclidean space cannot be used with categorical data that are defined on the nominal scale. For this to be feasible, the categorical attributes have to be binarized, i.e., a binary attribute needs to be created for each unique value of the original categorical attribute. Even with all categories binarized, experiments have shown that many clustering algorithms developed originally for continues-valued data do not handle such data too well.

1.2. Overview of our proposed solution

To discover clusters in categorical data, we propose a novel algorithm that is based on a simple genetic algorithm (GA). GAs, which have been developed to solve optimization problems, have recently been widely used in different areas and have been shown to be very successful [Tzafestas *et al.* 1999; and Chen 2002]. Here, we make use of GA's powerful ability to perform probabilistic search to try to find optimal cluster groupings.

Like other GA based algorithms, our clustering algorithm is as well based on GA operators; we need to decide on an appropriate encoding scheme and a suitable fitness function. For the encoding scheme, we choose to encode a cluster arrangement in each chromosome with each gene encoding a cluster. As for the fitness function, given a set of categorical data and the cluster arrangement as encoded in a chromosome, the fitness function we choose is probabilistic and is based on an information-theoretic entropy measure. It measures the frequency of occurrence of same attribute-value pair within each cluster so that a higher fitness

values means the records within the same cluster share more common attribute values. With this fitness function, we aim to optimize the "purity" of the attributes within a cluster.

The fitness measure that is used can be expressed as interesting patterns so that a set of attribute-value pairs can be correlated with a cluster label [Agrawal *et al.* 1993; Agrawal, Srikant 1994; and Agrawal *et al.* 1996]. For this, one may be attempted to use *support* and *confidence* measure that are used to determine if the pattern is interesting as a fitness measure. However, this requires users to define these interestingness measures and this is usually difficult to determine. To overcome this problem, the probabilistic measure the proposed algorithm employs does not require subjective thresholds to be provided. The interestingness of the pattern can be reflected by the value of the function we propose so that the fitter a chromosome is, the more interesting and meaningful the grouping of the chromosome encodes. And this is not affected by the users' choice of interestingness threshold.

In order to maximize the occurrence of an attribute value, it is necessary to regroup data records. This is achieved by special crossover and mutation operators that aim at swapping randomly selecting records from each cluster from one cluster to another. This process is repeated until there is no further improvement in the fitness of the best chromosomes.

In brief, the proposed GA based clustering algorithm has several useful features: (i) it uses an entropy based, rather than the distance based, similarity measure for clustering; (ii) since the fitness measure is probabilistic, it can be used even with noisy and missing values; (iii) it uses a reclassification technique to speed up the identification of optimal solutions; and (iv) it is able to express patterns discovered in each cluster explicitly to allow for them to be better interpreted. In addition to these features, it should be noted that the proposed algorithm could be modified to find a suitable number of clusters that should be discovered in a data set.

1.3. Outline of the Thesis

The thesis is organized as follows. In Chapter 2, a review of relevant literature is presented. We discuss how clustering can be performed with traditional clustering algorithms developed for pattern recognition or machine learning. In particular, we describe also some clustering algorithms developed specifically to handle categorical data. Besides, since GA has been used to perform clustering and our proposed algorithm is based on GA, in this same chapter, we also give a brief overview of how GA has been used in data clustering.

In Chapter 3, we describe our proposed GA-based clustering algorithm in details. We have evaluated the proposed clustering algorithm with synthetic and real data. In Chapter 4, we give the details of the experiments and discuss the results. Our proposed algorithm can be modified to allow the number of clusters to be discovered. In Chapter 5, we discuss the details of how our algorithm is modified to solve the problem. Finally, in Chapter 6, we give a summary of the work and a proposal for future.

Chapter 2

Literature Review

Research into clustering has traditionally concentrated on numerical data. Many effective algorithms have been developed for such data [Kaufman and Rousseeuw 1990; Ng and Han 1994; Zhang *et al.* 1996; Guha *et al.* 1999; Karpis *et al.* 1999; Ester *et al.* 1996; Ankerst *et al.* 1999; Wang *et al.* 1997; Sheikholeslami *et al.* 1998; and Agrawal *et al.* 1998 etc.]. When it comes to categorical data, however, not many clustering algorithms are developed specifically for them.

For many clustering algorithms, the Euclidean distance is usually used as a measure of similarity between two records. Unfortunately, this distance measure cannot be defined for categorical data. Recently, algorithms that measure data point-density have been developed for mining large numerical data sets for inherent clusters. However, density measures are also not very good measuring criteria for categorical data, as density is not a quality of such data.

For categorical data, a popular distance measure that can be used is the hamming distance measure. It is defined to be a count of the matches and mismatches of attribute values of two different objects or data records. Even with hamming distance, it should be noted that the concept of density and centroid is hard for some clustering algorithms to work effectively.

One approach to clustering categorical data is to ignore the distance measures altogether. Such an approach would make use of a probability measure instead. This

measure is based on counting the frequency of common features shared by records in the same cluster. The higher the frequency, the more similar the data records are in a cluster. In the following, we give an overview of existing clustering techniques that can be classified according to whether or not they are developed for continuous-valued data, or for categorical data. As our proposed algorithm is GA based, we also discuss how GAs have been used in clustering.

2.1. Continuous-Valued Data Clustering

As discussed above, many popular clustering algorithms employ a distance metric as a measure of how similar the records are. The most popular distance metric used is the Euclidean distance. Many optimization-based clustering algorithms, such as the *K*-mean algorithm, require users to specify the number of clusters ahead of time. One advantage of these techniques is that they are fast and easy to implement. Compared to optimization-based algorithms, hierarchical clustering algorithms do not require the number of clusters to be determined ahead of time. Such algorithms make use of dendrogram to identify similar records that should be grouped and to determine the number of clusters to form. Hierarchical clustering algorithms, compared to the optimization-based algorithms, can be slow.

2.1.1. Optimization-based Methods

Given *k*, the number of clusters to form, optimization-based methods attempt to partition a given data set into *k* clusters based on a criterion function [Han and Kamber 2001]. These algorithms seek to optimize a criterion function. The criterion function, often called a similarity function, can be a distance metric and is used as a partitioning function so that the data records within a cluster are “similar”, whereas the data records of different clusters are “dissimilar”.

However, to search globally for the optimised partitioning requires exhaustive enumeration of all possible partitions. This is computationally infeasible. For this reason, a locally optimised approach is usually used. This is done by trying to

optimise the local partitioning function like hill-climbing and producing successive clusters. It terminates when a local optimum is determined.

Among optimization-based clustering algorithms, the *K*-means algorithm is the most popular. It takes an input parameter *k* and partitions a given data set into *k* clusters by optimising a criterion function.

$$d(S,C) = \sum_{j=1}^{k} \sum_{s_q \in C_j} d(s_q, c_j)$$

where c_j is the centre of a cluster C_j, s_q is a record assigned to C_j, k is the number of clusters to be formed, and $d(s_q, c_j)$ is the Euclidean distance between data record s_q and c_j. Thus, the criterion function *d(S, C) is* attempting to minimize the distance of every data record from the centre of the cluster to which the record belongs.

More specifically, the *K*-means algorithm begins with the choosing of *k* records as the initial cluster centres. It then assigns each record to the cluster whose centre is the nearest to it. Once this is done for all records, the centre is recomputed for each cluster by calculating the cluster means. The records are relocated according to the updated cluster means until there are no changes.

There are quite a lot of variants of *K*-means clustering. They vary from the selection of the initial *K*-means and the criterion function, to the calculation of the cluster means. Instead of computing the means, the *K*-medoids algorithm [Kaufman and Rousseeuw 1990] used one record in the cluster as a representative. CLARA [Kaufman and Rousseeuw 1990] is a modified version of *K*-medoids based on sampling. Instead of finding representative records for the entire data set, CLARA draws a sample of the data set, applied *K*-medoids on the sample, and finds the medoids of the sample. The size of the samples depends on the number of clusters. Empirical results show that the size of $40+2k$, where k is the number of clusters to be formed [Kaufman and Rousseeuw 1990], gives the optimal result. CLARA will normally run several samples to select the best clustering arrangement.

Based on CLARA, an algorithm called CLARANS has been proposed [Ng and Han 1994]. CLARANS draws a sample of data records called nodes in each step of a search and this has the benefit of not limiting a search to a localised area. CLARANS takes two user parameters *maxneighbour* and *numlocal* while performing its tasks. *Maxneighbour* is the maximum number of the surrounding records of sample data record that are to be examined. *Numlocal* is the maximum number of local optima that can be collected. CLARANS begins by selecting a random data record. It then checks a sample of neighbour records of the data record, and if a better neighbour is found, it changes to the neighbour record as a local optimum and continues processing until the *maxneighbour* is reached. Otherwise, it declares the current record a local optimum and starts a new pass to search for other local optimum. After a specified number of local optimum, *numlocal* are collected, the algorithm returns the best of these local values as the medoid of the cluster.

2.1.2. Hierarchical Methods

Hierarchical clustering methods can be either agglomerative or divisive [Kaufman and Rousseeuw 1990]. Agglomerative methods, also known as bottom-up models, start with each record itself in the data set forming a single cluster. They then progressively join the nearest clusters into a larger cluster until all the clusters have merged into one. Divisive methods, also known as top-down models, are the opposite of the agglomerative methods. They start with all records forming a single cluster and, step-by-step, this single cluster is divided into smaller clusters. Ultimately, each record forms a single cluster.

These hierarchical clustering algorithms produce a tree of clusters, called a dendrogram, which shows how the clusters are related. It can also be a data structure as is used by BIRCH [Zhang *et al.* 1996]. The data structure, called a CF-tree (Clustering-Feature tree), allows records to be grouped from smaller clusters to form larger clusters hierarchically. Suppose the root of CF tree represents a large cluster containing all data records, the branches of the tree are subclusters of those records. A CF tree contains a measure called *clustering feature*. A *clustering feature* describes the number of data records in each subcluster, the linear sum and square

sum of distance among those records. A CF tree makes use of two parameters: branching factor *B* and threshold *T* to control the clustering of data records. The branching factor *B* specifies the maximum number of children per non-leaf node which stores sums of the *clustering features* of their children, and thus summarize *clustering feature* about their children. The threshold *T* specifies the maximum diameter of subclusters stored at the leaf nodes of the tree. These two parameters influence the resulting size of the tree [Han and Kamber 2001].

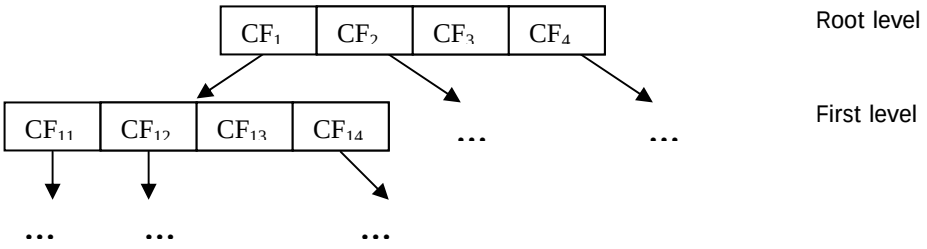


Figure 2-1. CF tree

The data records are incrementally inserted to the closest leaf node of the CF tree. Once inserted, it updates the *clustering features* from the leaf node up to the root node. If the diameter of the subcluster after insertion exceeds the threshold *T*, the leaf node and possibly other nodes are split. After all data records are inserted, the tree is rebuilt to improve the clustering quality by redistributing the data records again. The final clusters can be obtained by cutting the branches at a particular level.

Other than BIRCH, Chameleon [Karypis *et al.* 1999] is also a hierarchical clustering algorithm. It differs from other similar algorithms by its taking into account the dynamic model of clusters. Like other hierarchical clustering algorithms, it merges data records successively based on the idea of relative inter-connectivity and relative closeness. It merges clusters if their inter-connectivity and proximity are high enough with respect to the interconnectivity and closeness between records and the cluster. Chameleon first constructs a sparse graph for the data records according to their similarity. It then uses a graph partitioning method to partition the data records into a large number of smaller clusters. After that, it uses hierarchical method to progressively merge the smaller clusters into larger one [Han and Kamber 2001].

2.2. Categorical Data Clustering

There has not been much effort to develop clustering algorithms for categorical data. Among the more popular algorithms, the most well known one is called COBWEB [Fisher 1987]. COBWEB adopts a technique called concept learning. It groups data records with similar features together in one single cluster. Based on COBWEB, two variants of it have been developed. They include Autoclass [Cheesman and Stutz 1995] and ITERATE [Biswas *et al.* 1998].

Other conceptual clustering techniques [Michalski 1980; Fisher *et al.* 1991; Fisher 1987; Fisher *et al.* 1993; Anderson and Matessa 1991; Gennari *et al.* 1989; and Cheesman and Stutz 1995] developed by the machine learning researchers can also be considered for use in clustering records containing discrete-valued data. More recent work on clustering categorical data can be found in [Guha *et al.* 1998; Ganti *et al.* 1999; Huang 1997a; Huang 1997b; Huang 1998; Han *et al.* 1997; Gibson *et al.* 2000; Zhang *et al.* 2000; and Cristofor and Simovici 2002]. In the following, we briefly describe the three more popular approaches.

2.2.1. COBWEB

COBWEB [Fisher 1987] performs clustering by maximizing its ability to infer similarities between data records. COBWEB is a typical form of incremental conceptual clustering. It forms clusters of records that shared common concepts.

COBWEB performs hierarchical clustering in the form of a classification tree. It uses a heuristic measure called Category Utility to perform grouping. Categorical utility (CU) is a function for maximizing the similarity of records within a single cluster and the dissimilarity of records between two different clusters. CU makes use of two concepts: intra-cluster similarity and inter-cluster dissimilarity. Intra-cluster similarity is defined to be the probability $\Pr(A_i = a_{ij} \mid C_k)$ where A_i is the attribute i of a data record, a_{ij} is the value of attribute i and C_k is one of the k clusters. The larger the intra-cluster similarity within a cluster, the greater the proportion of data records

in that cluster that share this attribute-value pair, and the more predictable is the cluster that possess this attribute-value pair. Inter-cluster dissimilarity, on the other hand, is defined to be the probability $\Pr(C_k \mid A_i = a_{ij})$. The larger this value is, the fewer the data records in contrasting clusters that share this attribute-value pair, and the more predictable is this attribute-value pair contribute only to one cluster.

Given a dataset, COBWEB inserts records one-by-one into the classification tree by maximizing the category utility. The records are placed together under those nodes having the most attributes in common. The records traverse the classification tree several times till the tree become stable.

The placement of the records into clusters is highly sensitive to the input order. COBWEB solves this problem by introducing two operators that help make it less sensitive to input order. These operators are merging and splitting. When a record is inserted, the two best nodes are merged into a single node. Furthermore, COBWEB considers splitting the children of the best node among the existing tree. Both merging and splitting are based on category utility and these two operations provide a way for COBWEB to flexibly reallocate an assigned record.

2.2.2. The *K*-modes Algorithm

The *K*-modes algorithm [Huang 1997a; Huang 1997b; and Huang 1998] is a modified version of the *K*-means algorithm for handling categorical data. The *K*-modes algorithm, like *K*-means, takes an input parameter *k* and attempts to partition a given data set into *k* clusters by optimising a criterion function.

K-means identifies a representative of each cluster being formed by calculating the centre of all records in the cluster. However, the mean of two categorical attributes produces a meaningless value. For this reason, the mode of each attribute is used instead. More specially, the algorithm begins by choose *k* records as the initial cluster centres. It then assigns each record to a cluster whose centre is the nearest. The centre is then recomputed by calculating the cluster modes. To measure dissimilarity, *K*-modes define a simple dissimilarity matching function. The function calculate the

number of mismatch between the cluster centre C_j, where $j = 1, 2, \ldots, k$ and a record s_q with m attributes, where $i = 1, 2, \ldots, l$ and $n = 1, 2, \ldots, m$,:

$$d(S,C) = \sum_{j=1}^{k} \sum_{s_q \in C_j} \sum_{n=1}^{m} d(s_{q,n}, c_{j,n})$$

where

$$d(s_{q,n}, c_{j,n}) = \begin{cases} 0 & (s_{q,n} = c_{j,n}) \\ 1 & (s_{q,n} \neq c_{j,n}) \end{cases}$$

For each of the iterations, it tries to minimize the dissimilarity function. The lower the value of the dissimilarity function, the more the attributes the data record match with the cluster centre and the smaller the distance is between the data record and the cluster centre. For each assignment of the data record to a cluster, the mode is updated by finding the most frequent attribute value for that cluster. After all the records are assigned, the process is repeated. The records are considered for relocation according to the updated cluster mode until there are no further changes.

K-modes, like *K*-means, has the advantages of being computationally easy and has high scalability when processing large amount of and high dimensional data. *K*-modes are limited in their ability to find starting representatives or centres. Also, they are usually only able to converge to a locally optimal solution [Huang 1998].

2.2.3. Autoclass

The Autoclass [Cheeseman *et al.* 1988; and Cheeseman and Stutz 1995] system, developed by Peter Cheeseman et al., is an unsupervised Bayesian classification system that seeks for a maximum posterior probability classification.

Bayesian theory gives a mathematical calculus of the degrees of belief that a given sample belongs to a particular class/cluster. Let *s* be a data record whose cluster label is unknown. Let *H* be some hypothesis, such that the data record *s* belongs to a specified cluster *C*. Let *P(H|s)*, the probability that the hypothesis *H* holds given the observed data record *s*. More specifically, *P(H|s)* denotes the posterior probability of

H conditioned on *s*. It reflects the probability in *H* after seeing evidence *s*. In contrast, *P(H)* is the prior probability of *H*. It denotes that the probability *H* holds regardless of how the given data record looks. Given a data set with *k* clusters, C_1, C_2, …, C_k. Autoclass repeatedly creates a random assignment of the data records into any one cluster and then tries to massage this into a higher posterior probability of classification through local changes until it converges to some "local maximum". To find these local maximum, Autoclass uses the Expectation Maximization (EM) Algorithms [Dempster *et al.* 1977].

The EM algorithm is based on the fact that at a maximum, cluster parameters can be estimated from statistics. These statistics are obtained from the attributes of the data records belonging to that cluster and it is Bernoulli distributions for categorical attributes, Gaussian distribution for numeric attributes and Poisson distribution for number counts. Given the cluster parameters like cluster mean and cluster standard deviation, Autoclass computes a relative likelihood weight, w_{sc_k} that gives the probability that a particular record, *s* is a member of a cluster, C_k. Using these weights, cluster membership is expressed probabilistically rather than a specific assignment. Thus every one of the records in a given dataset is considered to have a probability that it belongs to each of the possible cluster. These cluster membership probabilities must sum to one for each of the data records. No crisp boundaries are presented among clusters. The records are assigned to the clusters with different degree of membership. The higher the membership, the more likely the records belong to that cluster.

The algorithm firstly computes the probabilistic cluster memberships of records using the cluster parameters. After the records are assigned, the cluster parameters are updated. It then reassigns those records according to the updated cluster parameters. It repeats until it stops changing. Then, it is said to find a local maximum. Autoclass can be stopped by a maximum duration, by a maximum iteration or by user intervention. When stopped, clusters can be obtained, each of which is described by a set of cluster parameters such as the mean and standard deviation of the attributes in the cluster. For example, "cluster A is described with height normally

distributed with mean 4.67ft and standard deviation 0.32 ft". A set of cluster weights further describes the probabilistic assignment of the records to each of the clusters. The main description of the Autoclass algorithm is depicted in Figure 2-2.

1. Determine the number of clusters, *k*, for a partition.
2. Randomly split the data into *k* groups and estimate the parameters of each cluster.
3. Repeat the redistribution process using a hill climbing method in an attempt to optimize the clustering structure, and
4. Adjust the number of *k*, and repeat steps 1-3.

Figure 2-2. Description of the Autoclass algorithm

From Figure 2-2, we can see Autoclass divided into two parts: (i) determining the number of clusters; and (ii) determining the parameters that optimize the most probable clusters. In determining the number of clusters, Autoclass suggests that including a cluster that has negligible posterior probability in the model cannot improve the likelihood of the better clustering. Similarly, the model in which a cluster has a negligible prior probability will always be less probable than models that simply omit that cluster. Autoclass begins with smaller number of clusters and searches to find the best cluster parameters for that number of clusters. When the resulting clusters have significant probability, the number of clusters is increased. This process repeated until the result has clusters with negligible posterior probability. The clusters with negligible posterior probability are removed and the optimal number of cluster is found.

2.3. Genetic Algorithm Clustering

Genetic algorithms demonstrate robust and domain-independent search characteristics. For this reason, genetic algorithms have been widely used in optimization-based clustering methods when the goal is to find a global optimum solution [Su and Chang 1998; Krishna and Murty 1999; Jiang and De Ma 1996; and Song *et al.* 1997].

Genetic algorithms can be used in clustering for purposes as varied as finding optimal numbers of cluster [Tseng and Yang 1997], initializing cluster centers [Kuncheva and Bezdek 1998; and Bezdek *et al.* 1994] and finding globally optimal clustering [Su and Chang 1998; Krishna and Murty 1999; Jiang and De Ma 1996; Song *et al.* 1997; Kemenade 1996; Kuncheva and Bezdek 1998; and Bezdek *et al.* 1994]. Recently, the focus of research has been on using GA to find clustering rules [Sarails *et al.* 2002] and using GA to find the number of clusters [Simovici *et al.* 2002; and Leon *et al.* 2006]. For example, GA has been used to acquire rules that identify dense regions and used an evaluation function to adjust those rules on coverage of the dense region [Sarails *et al.* 2002]. The identification of such rules also allows the identification of the attributes and ranges that contribute to the final cluster.

2.3.1. Genetic *K*-means Algorithm

The *K*-means algorithm must be the most typical of the optimization-based clustering algorithms. Although *K*-means algorithm is simple and computational efficient, it suffers from the problem of binding to local optimum. This is because the *K*-means algorithms search only those spaces that are close to their starting cluster centers. This search will be stagnant if the hill-climbing process has reached its local optimum. It cannot jump from one space to another space at random.

In [Krishna and Murty 1999], it attempts to solve this problem by stochastically searching from a different solution space. It attempts to find optimal initial centers for *K*-means. The algorithm starts from randomly generate initial centers. *K*-means is then run against these initial centers to find the clusters. The cluster centers with the least within-group sum of distance are chosen for genetic operations and are picked up as initial centers for next generation. The GA process stops when the maximum number of generation is reached.

2.3.2. Genetic rule-based Algorithm

A genetic rule-based clustering algorithm [Sarails *et al.* 2002] presents a tool for clustering using a rule-based GA. For a given data set, the algorithm tries to find a set of clustering rules. The number of rules corresponds to the number of desired clusters. Each rule contains *d* genes, where each gene corresponds to an interval involving one feature. Each gene belongs to a rule contains two fields: a lower boundary (lb_i) and an upper boundary (ub_i), where lb_i and ub_i denotes the lower and upper values of the i^{th} feature of the rule. The lower boundary and the upper boundary are used as comparators for a selected feature. The comparison operator may be ≥ (larger or equal) and ≤ (smaller or equal). For example, given a dataset containing data records that have two attributes, tax and salary, we can define two clusters as follows:

Cluster 1:

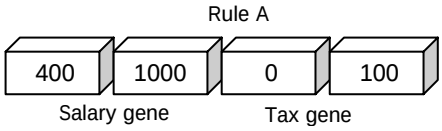
Rule A: ((400 ≤ salary ≤ 1000) && (0 ≤ tax ≤ 100))

Cluster 2:

Rule B: ((0 ≤ salary ≤ 200) && (300 ≤ tax ≤ 400))

The entire chromosome will appear as in Fig 2-3.

Chromosome A:



Chromosome B:

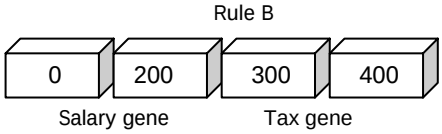


Figure 2-3. Grouping rules encoded in chromosomes

The genetic rule-based clustering algorithm evaluates chromosome fitness along five parameters: rule asymmetry, rule density, rule coverage, rule homogeny, and degree

of rule overlap. Each criterion has a role in maximizing interclass dissimilarity and intraclass similarity. For a data record belonging to one particular rule, rule asymmetry ensures uniform distribution of data records relative to the rule centre. Rule asymmetry takes two measures, *Cr* and *Cp*. *Cr* is the rule centre. For example, given a rule with 400 as its lower and 1000 as its upper bound, *Cr* would be 400+1000 / 2 = 700. *Cp* measures the average of all points in a cluster. Then, it calculates how much *Cp* deviate from *Cr*. Rule density measures the number of data records in a unit area. Rule coverage measures how many data records are covered by a rule. Rule homogeny ensures that the data records within a cluster are close to each other and penalises where there are holes in dimensions. Rule overlap is used to penalise duplicate rules over generations.

The higher the score in the five parameters, the best is the chromosome. The fittest chromosomes are then selected, crossover mutated and carried to the next generation. The genetic algorithm terminates after it reaches the maximum iterations. The best chromosome is the result of the final clusters. The experimental result shows that it can identify clusters of various shapes, sizes and densities [Sarails *et al.* 2002].

2.3.3. An Information-Theoretical Approach to Genetic Algorithms for Clustering

An information-theoretical approach to genetic algorithm for clustering has been proposed recently [Cristofor and Simovici 2002; Simovici *et al.* 2000; and Simovici *et al.* 2002]. Given a data set with categorical attributes, the algorithm uses the entropy to measure the similarity of data records within each cluster. The similarity can be measured by using Shannon and Gini entropy [Cristofor and Simovici 2002]. The Shannon entropy, the uncertainty within a cluster, decreases when the attributes within the data records are similar. On the other hand, the entropy increases when the data records within a cluster are dissimilar. From this perspective, it attempts to construct clusters that the sum of the dissimilarities among the data records is minimal.

In order to search more efficiently the space of possible clustering, genetic algorithm is employed. The clustering process is divided into two phases. In the first phase, it estimates the influence of each attribute on the decision to place a record in a specific cluster. This estimation can be obtained from a domain expert or by using a set of training records. In the second phase, it searches for a clustering of the entire set of records such that the attributes of the records influence the clustering to the extent obtained in the first phase. The partition of the records is based on an entropy measure that is called impurity measure by the algorithm to calculate the similarity of records within a cluster. The definition of impurity is depicted in Figure 2-4 [Simovici *et al.* 2002] as follow:

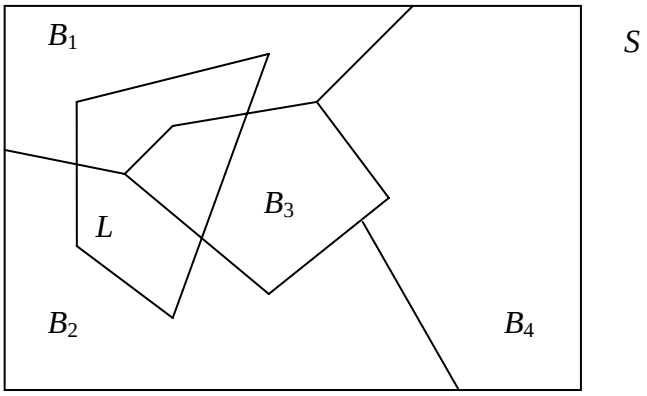


Figure 2-4. A partition of S and an impure subset L

Let *f* be a function, *S* be a set and let $\pi = \{B_1,...,B_n\}$ be one partition of *S* and *L* be a subset of *S*. The impurity of a subset *L* of *S* relation to a partition π is the impurity generated by *f* as follow [Simovici *et al.* 2002]:

$$imp_{\pi}^{f}(L) = f\left(\frac{|L \cap B_1|}{|L|}\right) + ... + f\left(\frac{|L \cap B_n|}{|L|}\right)$$

The impurity measure generated by *f* can be a Shannon's entropy measure or a gini index. For Shannon's entropy $f_{ent}(p) = -p \log p$ [Simovici *et al.* 2002] and for gini, $f_{gini}(p) = p - p^2$ [Simovici *et al.* 2002], or

$$f_{peak}(p) = \begin{cases} p & \text{if } 0 \le p \le 0.5 \\ 1-p & \text{if } 0.5 < p \le 1 \end{cases}$$

When the subset L is included in one of the blocks of partition π, $imp_{\pi}^{f}(L) = 0$ and L is called π*-pure* set; otherwise, L is called π*-impure.*

If $\sigma = \{L_1,...,L_m\}$ be the partition of the cluster for the data set, the quality of a clustering is measured as:

$$H^{f}(\pi \mid \sigma) = \sum_{j=1}^{m} \frac{|L_j|}{|R|} imp_{\pi}^{f}(L_j) = \frac{1}{|R|} \sum_{j=1}^{m} |L_j| \sum_{i=1}^{n} f\left(\frac{|B_i \cap L_j|}{|L_j|} \right)$$

where R is the number of records in data set [Simovici *et al.* 2002].

This quantity is called the conditional entropy of π relative to σ. In other words, the conditional entropy $H^{f}(\pi \mid \sigma)$ is the average value of the specific impurity of the classes of the partition σ relative to the partition π. The dissimilarity d [Simovici *et al.* 2002] is defined as

$$d^{f}(\pi,\sigma) = H^{f}(\pi \mid \sigma) + H^{f}(\sigma \mid \pi)$$

When π is close to σ, meaning that their classes have many attribute-values pair in common, then both $H^{f}(\pi \mid \sigma)$ and $H^{f}(\sigma \mid \pi)$ are close to 0, so $d^{f}(\pi,\sigma)$ is close to 0 [Simovici *et al.* 2002].

A genetic algorithm is employed to find such partition and $d^{f}(\pi,\sigma)$ is used as the fitness function. Each record is randomly assigned with a cluster label and the fitness of a chromosome is calculated by the above notion. At the end of each generation, the fittest chromosome amongst all will be brought forward to the next generation until some stopping criteria is reached. Since unwanted clusters will increase the impurity of the partition, by varying the number of partitions, the number of natural clusters existing in the data may be discovered.

2.3.4. ECSAGO

ECSAGO [Leon *et al.* 2006] is a genetic algorithm for clustering based on Hybrid Adaptive Evolutionary (HAEA) algorithms and Unsupervised Niche Clustering (UNC). HAEA is a parameter adaptation technique that automatically learns the rate of genetic operators. In each generation, one genetic operator such as crossover and mutation is selected for each chromosome according to dynamically learned operator rates that are encoded in the chromosome. If the genetic operator chosen requires mating, it will pick another chromosome to finish the operation. Unsupervised Niche Clustering (UNC) is a clustering algorithm that based on a niche technique. A niche technique only allow crossover where the two parents are in the same niche in order to prevent bad offspring. To determine clusters from the two parents are in the same niche, we should make sure the two parents are close enough, within a distance known as niche radius. If the distance between two clusters centre from two parents are within the niche radius, the two parents can perform crossover. Genetic algorithm maintains a population pool where chromosomes may contain different number of clusters. These chromosomes are in their best fitness that they are said to reach their local optimum. As a result, the final population maintain a pool of chromosomes with different number of clusters. The chromosome is ranked according to the fitness function and the best chromosome is picked up. The number of clusters and the cluster grouping can then be determined. The fitness function adopts the Euclidean distance measure where distance between records within the same cluster should be minimized. This technique provides benefit that it can locate all optima of multi-modal problem such as finding all optimal clustering for different number of clusters in a single course of action.

Chapter 3

A GA-based approach to clustering categorical data

In this Chapter, we describe a GA-based algorithm for the clustering of categorical data. Specifically, we describe (i) how different clustering arrangements can be encoded in a population of chromosomes, (ii) a fitness function that can allow the interestingness of different clustering arrangements to be compared; (iii) a crossover operator that can allow interesting patterns discovered in different clustering arrangements to be exchanged; (iv) a mutation operator that can allow variations to clustering arrangements so as to avoid local minimum and (v) a reclassification operator that can correct the wrong clustered records into correct one.

In Section 3.1, we give a formal definition of the clustering problem we are tackling and we also introduce the notations we will be using in describing the proposed algorithm. For the purpose of compare-and-contrast, we describe, in Section 3.2, popular proximity measures used in traditional clustering algorithms. Before we present our proposed algorithm, we first introduce a simple GA that our algorithm is based on in Section 3.3. In Section 3.4, we describe the details of our GA-based clustering algorithm. In particular, we describe a scheme for encoding different clustering arrangement in chromosomes. We also describe the genetic operators we use in the proposed algorithm for crossover and mutation. In Section 3.5, we introduce a fitness function for the evaluation of chromosomes.

3.1. The Clustering Problem and Notations

The clustering problem, in its simplest form, can be considered as a special kind of partitioning problem. The problem of partitioning can be defined to be the partitioning of a data set with records, *S*, into a collection of mutually disjoint subset s_i of *S* such that $s_i \neq \varnothing, i = 1,\ldots,m$,

$$\bigcup_{i=1}^{m} s_i = S$$

and

$$s_i \cap s_j = \varnothing \text{ where } i \neq j, i, j = 1,\ldots,m$$

Many well-known problems like bin packing (assigning item to bins), graph coloring (assigning specific colors to nodes of a graph) and clustering (grouping of similar objects) etc., can also be classified into partitioning problems [Falkenauer 1998]. For these problems, when the number of objects and the attributes increase, the number of possible partitions that need to be considered will explode drastically [Theodoridis and Koutroumbas 1999] and there is no practical way to check for every possible combination. Traditionally, these problems are solved by the use of different heuristics including simulated annealing, genetic algorithm, evolutionary programming and tabu search are developed under such circumstances [Falkenauer 1998].

The clustering problem can be described as follows. Given a data set, *S* containing *N* records (or objects), s_1, ..., s_q, ..., s_N, with $q = 1, \ldots, N$. Suppose that each record in the data is described by *n* distinct attributes, A_1, ..., A_j, ..., A_n, where A_j can take on continuous, or discrete data values and their respective domain of values are denoted as $dom(A_1), \ldots, dom(A_j), \ldots, dom(A_n)$ where $dom(A_i)=\{a_{i1}, \ldots, a_{ij, \ldots,} a_{im}\}$. Suppose also that the data are noisy in the sense that a certain percentage of values are incomplete, inconsistent or incorrect so that the patterns inherent in the data are not completely deterministic. The clustering problem that we are concerned with is to discover a meaningful grouping of the data in, *S*, so that data records that share common characteristics are grouped together into the same group and data records

that are different in characteristics are placed separately. In other words, we need to discover a $C_k = \{s_q \mid q \in \{1, \ldots, N\}\}$, $k = 1, \ldots, K$, where K is the total number of discovered clusters and $\bigcup_{k=1}^{K} C_k = \{s_1, s_2, \ldots, s_N\}$ and $C_k \cap C_l = \varnothing$, $k \neq l$.

In grouping records into groups that share some common characteristics, it should be noted that, if one is to perform clustering by trial-and-error, the number of possible groupings that need to be considered could be substantially large. This is a combinatorial optimization problem that Genetic Algorithms (GA) can tackle well. In fact, GA has been used for clustering as well as in other data mining problems [De Jong 1988; Chan and Au 2001; and Sarails *et al.* 2002]. The benefit of GA is that it searches a solution space probabilistically at different places and is better able to avoid local maxima. Also, GA can be implemented in parallel and can perform its tasks relatively efficiently when handling very complex problems.

Given the above notation, it should be noted that the clustering problem is therefore to find a mapping f where $f : S \rightarrow \{C_1, \ldots, C_k\}$and a cluster, C_j, contains precisely those records or objects mapped to it [Dunham 2003].

How good f is can be evaluated with a goodness measure or cost function that can differ from applications to applications if domain-specific knowledge is known. In general, however, a clustering arrangement is good if all records in a cluster share the same characteristics and all those in the other clusters are characterized differently. The greater the homogeneity within a group and the greater the difference between groups, therefore, the better is the cluster arrangement.

Since the goal of clustering is to group similar records, the concept of similarity and what an adequate degree of similarity is needed to be properly defined [Steinbach *et al.* 2001]. The use of different similarity measures may lead to different clustering arrangement and different total numbers of clusters. Hence, it is important that such measures be properly defined [Fraley and Raftery 1998; Steinbach *et al.* 2001; and Berkhin 2002].

3.2. Similarity measures

Since clustering is concerned with the grouping of similar records and separating of dissimilar records, we need to define a proximity (or a similarity) measure. This measure can be denoted as *proxim*(s_i, s_j) and its magnitude should reflect how similar two records s_i and s_j are [Theodoridis and Kourthoumbas 1999; Everitt *et al.* 2001; and Dunham 2003]. Often, the definition of similarity is the crucial factor in cluster analysis in data mining. But unfortunately, it is not easy to formulate a universally applicable definition for *proxim*(s_i, s_j) as there are different types of data, binary, nominal, ordinal, and ratio (or continuous data), etc., and each data type may need to be taken into consideration so that they may be measured differently if needed.

For the purposes of proximity measurement, data may be categorised as either numerical or categorical. For numerical data, the most often used proximity measure is "distance" which can be denoted as *dist*(s_i, s_j) or δ_{ij}. A distance metric must satisfy the triangular equality: $\delta_{ij} + \delta_{jk} \geq \delta_{ik}$, i.e., the sum of the distances between s_i and s_j and s_j and s_k must be greater than or equal to that of the distance between s_i and s_k.

A record is considered to belong to a cluster if its distance with respect to that cluster is smaller than those of other clusters. A variety of distance measures have been proposed for distance-based clustering. They include Euclidean distance, Manhattan distance, Chebyshev distance, Pearson correlation and angular separation. Of these distance measures, the most commonly used one is the Euclidean distance measure. The Euclidean distance between two records s_i and s_j are defined as:

$$dist(s_i, s_j) = \left\{ \sum_{k=1}^{p} (a_{ik} - a_{jk})^2 \right\}^{1/2}$$

where a_{ik} and a_{jk} are the kth attribute value of the records s_i and s_j respectively.

Since attributes may be measured with different scales and if the Euclidean distance is used, it could result in unfair weighting of the importance of some attributes. For continuous-valued attributes, a step called normalization is usually required before distances are computed.

For data that is categorical, they can be further classified into either binary or categorical attribute. A binary attribute has only two states: 0 or 1, *or* true or false, where 0 means that the attribute is absent, and 1 means that it is present. Binary attributes can be considered as categorical attributes that take on two different values.

If spatial distance is applied to discrete data, it should be noted that it would lead to invalid measurements and misleading clustering results [Guha *et al.* 1999]. To deal with discrete-valued attributes, they are typically converted into binary variables first so that some similarity measures can apply. These measures include, for example, the simple matching coefficient, and the Jaccard coefficient [Everitt *et al.* 2001]. The simple matching coefficient is given as in Table 3-1 and can be represented as follows:

$$sim(s_i, s_j) = (a + d)/(a + b + c + d)$$

where a denotes the number of binary attributes that are positive for both records, d denotes the number of binary attributes that are negative for both records, c denotes the number of binary attributes that are positive for s_i and negative for s_j and b denotes the number of binary attributes that are negative for s_i and positive for s_j.

It should be noted that a binary attribute may contain values that not of equal importance. For example, 1 may be considered as much more significant than 0. A similarity measure defined to tackle this kind of attribute is called non-invariant similarity. The most well-known such similarity is the Jaccard coefficient which is defined as

$$sim(i, j) = (b + c)/(a + b + c)$$

where the parameters, a, b and c are the same as denoted above.

		Object j		
		1	*0*	*Sum*
Object i	*1*	*a*	*B*	*a + b*
	0	*c*	*D*	*c + d*
	Sum	*a + c*	*b + d*	*p = a + b + c + d*

Table 3-1. A contingency table for binary variables.

A categorical attribute is an attribute that can take on more than two values other than 0 or 1 as is with binary attributes. For example, attributes such as *colour*, *weather*, etc. Normally, they can take on values other than 1 or 0. Even if it does happen to take on a value of 1 or 0, it does not need to be understood as suggesting a binary relationship. A categorical attribute can be dealt with in the same way as a binary attribute, with each value being regarded as a single binary attribute. The dissimilarity between two records s_i and s_j can then be computed as

$$sim(s_i, s_j) = \sum_{q=1}^{n} \delta(a_{iq}, s_{jq})$$

where n is the number of attributes in s_i and s_j [Huang 1998; and Han and Kamber 2001]

$$\delta(a_{iq}, a_{jq}) \equiv \begin{cases} 0 & (a_{iq} = a_{jq}) \\ 1 & (a_{iq} \neq a_{jq}) \end{cases}$$

Other than the above measure, probabilistic similarity measures have also been proposed for categorical attributes [Good 1966]. Such measures are based on the idea that a group of records that group together should share a common set of features.

Even though they are developed to handle data of different types, it should be noted that they are normally used to measure pair-wise distances. More global information, such as information regarding what the main attributes are that characterize a particular cluster cannot usually be obtained during the clustering process. For more effective clustering, it would be useful for such information to be considered for the cluster formation process. Towards this goal, we propose a GA based clustering which we will describe in the following sections.

3.3. A Simple Genetic Algorithm

The simple GA was first proposed by Holland in 1975, forms the basis of all extended ones. In a simple GA, a set of solutions is mapped to a set of representations in a representation space. These representations are encoded into a string form that is called a chromosome. The most typical form of chromosome is a string of bits (e.g. 1011, 1101), but in different applications the representations can be encoded differently, as, for example, integers, strings, or symbols.

Given the chromosomes, the simple GA is then set to carry out a parallel search in the representation space. At each search, a fixed number of representations are examined and those which best fit the solution are kept. After a number of searches, only good solutions remain and the best of these is taken as the ultimate best solution.

The mechanism by which the best representation can be found and retained is the result of the GA applying a series of genetic operators. These operators are of two types, evolving operators and survival operators. Evolving operators such as crossover and mutation operators are responsible for evolving representations to better solution. Survival operators such as selection and replacement operators are responsible for maintaining the best solution in the population. It is these best representations that the evolving operators operate on, so the population can gradually evolve. The mechanism of the simple GA is shown in Figure 3-1 below.

A simple GA uses a generational approach for chromosome replacement. In a generational approach, all chromosomes will be considered for reproduction. All chromosomes in the previous generation will be replaced by the siblings producing in the new generation, i.e. in a generation with a population of 50 chromosomes, any 50 of them may be selected, mate and reproduce 50 new chromosomes. The old 50 chromosomes will be replaced by the new 50 chromosomes that will again reproduce.

1. Initialise a population of chromosome.
2. Evaluate each chromosome by fitness function.
3. Select a pair of chromosomes from the population.
4. Crossover every pair of chromosome and reproduce 2 new child chromosomes.
5. Mutate the child chromosomes.
6. Delete the old generation of chromosomes to accommodate the new chromosomes and this produce the new generation.
7. Repeat Step 3-7 until reaching some termination criteria.
8. Return the best chromosome.

Figure 3-1. Convention genetic algorithm

During reproduction, two operations – crossover and mutation – are usually used. For crossover, part of the chromosome from one of the parents will be combined with a part from another. A typical simple GA usually employs a single-point crossover, i.e, a cut-point is randomly selected in both chromosomes and the two parent chromosomes involved can exchange their alleles across the cut-point to form a new pair of chromosomes.

After crossover, a mutation operator operates on the newly formed chromosomes. For mutation, certain alleles of a chromosome are randomly selected. If the representation or encoding of a chromosome is in binary form, then the selected alleles will be swapped from 0 to 1 or 1 to 0. When the encoding is in symbolic format, mutation will change the current symbol randomly to other symbols.

The GA as proposed by Holland has certain problems such as duplication of chromosomes and loss of the best chromosomes across generations. In the past several decades, researchers have attempted to improve it in order to yield better performance. For example, in [Whitley and Kauth 1988], a difference replacement

mechanism than generational replacement, called GENITOR, has been proposed. It has been called steady-state genetic algorithm (SSGA) in [Syswerda 1989].

In generational replacement, many best chromosomes cannot be kept and their genes may be lost in the next generation. The outcomes from a traditional simple GA can therefore be undesirable. In SSGA, however, only a small portion of the current population will be replaced by new chromosomes. The remaining chromosomes will remain intact and will stay in the next generation. For SSGA, we need to specify the number of newly created chromosome to be inserted into the current population. Usually, this parameter should be small when compared to the entire population. In fact, in a typical SSGA, only one or two chromosome will be chosen for replacement at a time.

Since the number of chromosomes in the old population to be removed and the number of chromosomes in the new population to be inserted are decided by the users, it should be noted that generational replacement could actually be viewed as a special case of SSGA in which the number of replacement equals the size of the population.

Compared to generational replacement, SSGA can guarantee to keep the best chromosome discovered so far. However, SSGA takes longer time to converge and it also suffers from the duplication problem as mentioned earlier that can cause premature termination of the evolutionary process. To overcome this problem, it is proposed that an “SSGA without duplicates” [Davis 1991] approach be adopted. This approach discards children that are duplicates of current chromosomes in the population, i.e., they are not inserted into the current population. Experiment shows that the performance for “SSGA without duplicates” is better than that of SSGA and the simple GA that performs generational replacement [Davis 1991].

The merits of GA as an efficient technique for searching for optima in very large search spaces builds on the foundation that it can limit and confine its search scope on several dimensions while keeping the remaining dimension intact. The success

relies on the selection scheme. Consider a population of chromosomes, during generations, those chromosomes with the least scores are eliminated while those with the highest scores remain in the pool. After a number of generations, an evolutionary process tend to converge and one may find that most segments of the chromosomes in a population may be the same (see Figure 3-2),

4 Chromosome with point 1 – 5 in commons:

Chromosome 1: 10110111 Chromosome 2: 10110000

Chromosome 3: 10110011 Chromosome 4: 10110101

Crossover in Chromosome 1 and 3 at point 6:

Chromosome 1: 101101|11 Chromosome 2: 101100|00

Chromosome 1: 101101|00 Chromosome 2: 101100|11

Newly formed chromosome still with point 1 – 5 in commons.

Figure 3-2. Four chromosomes in a population

This is because, for crossover to apply to two randomly selected chromosomes, the common segment will still remain in common. There is no method in crossover to change the allele of the common parts and these common genes can be represented by a “don’t care” symbol * as Figure 3-3.

Chromosome 1: *****111 Chromosome 2: *****000

Chromosome 3: *****011 Chromosome 4: *****101

Figure 3-3. Four chromosomes represented with “don’t care” symbols

For the chromosomes as shown, a new combination of the values for the last three genes can be produced. If there is no means for crossover to change the common segments, a GA may quickly converge to its local optima. To overcome the problem, a GA has to make use of mutation.

For the mutation operator, a gene in a chromosome is randomly selected and its value replaced with a randomly selected value from the list of possible ones. With mutation, there can be a chance to introduce variation in the common segment. The chromosome with the mutated gene can be evaluated and will remain in the next generation if it is good enough.

Depending on a mutation rate set by the users, great changes can be made to a population. However, in general, mutation rate are set very small and mutation normally changes only very few genes. In fact, mutation serves the purpose to keep most genes intact in the chromosome. Increase in mutation rate may result in the best genes disappearing in a particular generation thereby resulting in much longer time to locate optimal solutions.

3.4. A GA-Based Clustering Algorithm for Known K

The proposed GA, which we call GACDD (GA Clustering for Discrete-valued Data), is modified from the simple GA in several ways to suit the clustering task. First, it uses a Steady-State (SS) without duplicate replacement scheme during reproduction to make sure that only the better solutions are passed through generations. Second, the encoding of chromosome is modified for the handling of discrete data clustering. The cluster labels are encoded into the genes instead of the record labels. When clusters, rather than individual records, are taken as building blocks, we can maintain the best cluster groupings during evolution. Third, instead of swapping records, the crossover operator we used swaps clusters between two parent chromosomes. Similarly, the mutation operator operates on a cluster. As for selection, the *tournament selection* [Falkenauer 1998] is used for the proposed GA. It is chosen so that better chromosomes can have a better chance for selection.

For the fitness function, a *weight of evidence* measure is defined and used as a calculation for the fitness of a chromosome. The *weight of evidence* makes use, in turn, of a probabilistic information calculation defined on the concept of *entropy*. It is computed using an algorithm that can uncover interesting relationship between

different attributes. These relationships can be interpreted as association relationships with different support and confidence [Agrawal *et al.* 1993; Agrawal and Srikant 1994; and Agrawal *et al.* 1996]. By the use of a statistical measure, we can identify only those interesting relationships that are statistically significant. Having found all these relationships, we can then calculate their information entropy in terms of weight of evidence. In brief, the proposed GA based clustering algorithm can be described in several steps as given in Figure 3-4:

ALGORITHM 3.1. GACLUSTERINGFORDISCRETEDATA (GACDD)

```
1.  Let CR be crossover rate
2.      MR be mutation rate
3.      P be population
4.      M be number of chromosomes in the population
5.      N be number of records
6.  Set t = 0
7.  Initialize(P_t)
8.  Encode(P_t)
9.  Begin
10.     Repeat
11.         Fitness-Function(P_t)
12.
13.         # Selection, crossover, mutation and reclassify
14.         Begin
15.             For each i to (CR * P) / 2
16.                 Parent_1 = Select(P_t)
17.                 Parent_2 = Select(P_t)
18.                 Child_1, Child_2 = Crossover(Parent_1, Parent_2)
19.
20.                 Begin
21.                     For each i to (MR * N)
22.                         Mutate(Child_1)
23.                         Mutate(Child_2)
24.                     End for each
25.                 End begin
26.
27.                 Begin
28.                     For each i to M
29.                         Reclassify(P_t)
30.                     End for each
31.                 End begin
32.
33.                 Children-Array ← Child_1, Child_2
34.             End for each
35.         End begin
36.         P_t+1 = Steady-State-Replacement(Children-Array, P_t)
37.
```

38. **Until**(**Termination condition(s) =** true)
39. **End begin**

Figure 3-4. Algorithm of SSGA for our proposed clustering algorithm

3.4.1. Encoding Scheme

One way to encode a clustering arrangement in a chromosome is to encode the cluster label of a record in each gene, i.e., if we have *N* data records, we will have a chromosome of *N* genes. However, as pointed out in [Falkenauer 1998], the use of this representation scheme can result in duplicate chromosomes. For example, if a data set of six records is to be grouped into two clusters represented as 1 and 0, then the two chromosomes encoded as [000111] and [111000] will have exactly the same grouping of data records, i.e., the first three records being in one group and the second three being in another.

While this will not prevent a GA from finding an optimal solution, the presence of duplicated chromosomes does greatly affect the algorithm's search efficiency since (i) it reduces the diversity of the population and this essentially lowers the crossover rate, (ii) computational resources are wasted, as the reproductive processes do not always result in different chromosomes.

To overcome these problems, we propose an encoding scheme that can better facilitate the discovering of a more "optimal" clustering arrangement. In doing so, we took into consideration the followings: (i) the representation space should be complete, that is, all possible solutions must be representable in the space, otherwise we may never be able to find an globally optimal solution, (ii) the mapping from the solution space to the representation space should be one-to-one, i.e., one solution can only be mapped into one representation and it cannot be mapped into two or more different representations and vice versa so as to avoid redundancy. With these points in mind, we propose to encode each cluster in a gene in a chromosome so that each gene contains a number of record labels.

Using the notations as described above, for example, we can represent a data set, S of N = 10 records as s_1, ..., s_q, ..., s_{10}. Assume that we would like to encode the clustering arrangement consisting of 3 clusters, C_1, C_2 and C_3. Assume further that C_1 consists of three members, s_1, s_5, s_6, C_2 consists of three members, s_2, s_4, s_{10}, C_3 consists of four members, s_3, s_7, s_8, s_9. This cluster arrangement of each chromosome can be represented and implemented using a *list-of-list* structure as shown in Figure 3-5 below.

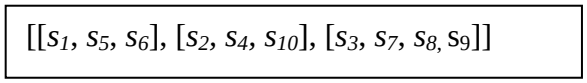


Figure 3-5. A list-of-list representation of a chromosome

This representation scheme used by GACDD has the advantage that it can represent all possible clustering arrangement without any inconsistency. In addition, it can avoid the problem of two chromosomes encoding exactly the same clustering arrangement.

3.4.2. Initialization

During initialization, for a data set that is to be divided into K clusters, we will need to randomly generate K genes, $C_k = \{s_q \mid q \in \{1, 2, \ldots, N\}\}$, $k = 1, 2, \ldots, K$ and $\bigcup_{k=1}^{K} C_k = \{s_1, s_2, \ldots, s_n\}$ and $C_k \cap C_l = \varnothing$, $k \neq l$. Given this requirement, GACDD initializes the population randomly using a random number generator as follows. For each record, s_q, $q \in \{1, 2, \ldots, N\}$, it is uniformly randomly assigned to one of C_1, C_2,…, C_K. This assignment process is done with a random number generator that generates numbers from 1, …, K. A record is considered to be a member of C_k, if the number generated for it is k. Once all the records are randomly assigned to a cluster based on the number generated for them, they are organized into the *list-of-list* structure as discuss above. For a population P containing M chromosomes, the population need to be initialized M times.

For clarification, pseudo code is specified for each function. The symbols in the pseudo code are defined as follows: P_t is the population of the t^{th} generation. $P_t[i]$ is the i^{th} chromosome in the population of the t^{th} generation which also denote as *list*. A *list* consists of K lists representing K clusters where K is the predefined number of clusters. Hence, it results in a *list-of-list* structure. *list*[k] is the k^{th} list of the list representing the k^{th} cluster. S is the record set. $S[q]$ is the q^{th} record and s is a single record of the data set. The initialization algorithm is summarized as follows in Figure 3-6.

```
1  proc initialize(↔P_t, ←S)      /* P_t is population and S is record set */
2      for i = 1 to |P_t| do       /* For each chromosome in the population */
3          P_t [i] = set_list (K)  /* Initilize the list of list to contain K list */
4          /* Assign each record to the lists */
5          for q = 1 to |S| do k = rand(1 …K); add(S[q], P_t [i], k) od
6      od
7
8  proc add(←s, ↔list,←k)
9      add_record(list[k], s)      /* Add record S[q] to list[k] */
```

Figure 3-6. Pseudo code for Initialization

3.4.3. Selection

To select chromosomes for reproduction, GACDD makes use of the tournament selection mechanism. Roulette wheel or proportional selection is the most popular selection mechanism in GA applications. However, it is sometimes not the best approach to be used. First, it exerts high selective pressure on the best chromosome [Falkenauer 1998]. For a population with members having very large differences in fitness values, the best chromosome is often chosen much more often than the others if the proportional selection mechanism is used. This may result in rapid convergence to local minimum. This mechanism also does not work too well when the fitness of a chromosome may be negative. This is because, for proportional selection, we need to compute a relative fitness with respect to the minimum fitness in the population. The usual approach to do so is to set the minimum value to zero, subtract the fitness of each chromosome from this baseline and re-calculate the relative fitness of the chromosome. However, as converting negative fitness to a positive number using

such an approach does not reflect the natural fitness of chromosomes, this is not a preferable.

Instead of proportional selection, the *ranking selection* mechanism can be considered. Under such a mechanism, all chromosomes are ranked according to their fitness values. Once this is done, the proportional selection mechanism can be used on the ranked chromosomes. Again, this practice may not truly reveal the fitness though the problem of selective pressure is avoided [Falkenauer 1998].

Since GACDD employs a fitness function called the *weight of evidence* (details are described later in this chapter) function and that this function can take on negative values and since there can be relatively large differences in fitness values, the use of the proportional selection or ranking selection mechanisms may lead to premature convergence. For this reason, we choose the tournament selection mechanism as it can better avoid these undesirable characteristics.

Tournament selection works as follow. For each selection, we randomly select two chromosomes in the population. We then compare their fitness. The fitter one wins and will be picked up for further operation. The loser will be put back to the pool of chromosomes. The selection repeats until we have taken sufficient chromosomes. The selected chromosomes will pair up, undergo crossover and form new chromosomes for the next generation. Tournament selection performed well with steady state genetic algorithm [Falkenauer 1998].

```
1  proc select(←P_t, →list)
2      index1 = rand(1…|P_t|); index2 = rand(1…|P_t|);
3          /* Calculate the fitness of the chromosomes */
4          if (get_fitness(P_t [index1]) > get_fitness(P_t [index2]) then
5              list = P_t [index1]; return list;
6          else list = P_t [index2]; return list; fi
```

Figure 3-7. Pseudo code for Selection

3.4.4. Crossover

With the encoding scheme described above, different genes in the chromosome encode different clusters and since different clusters can have different number of records, these genes are of different length. To ensure that the grouping arrangement of a chromosome be exchanged with another can evolve to better one, we must ensure only a small part of the chromosome is modified every time during crossover. Besides, crossover should happen at the cluster edge. If the crossover cut-off point is inside the cluster instead of the cluster edge, a chromosome will be cut and then combined independently of each other. For this reason, it is hard to allow good solutions to survive. When the population reaches an optimal, the crossover operator may destroy the optimal solution. To overcome this problem, in GACDD, the crossover operator is chosen in such a way that an entire cluster is selected randomly and exchanged between two chromosomes during reproduction. The details of the crossover procedure are given as follows.

After two parents are selected for crossover, a gene, $C_{k,}$ is then randomly selected from the first parent. This gene is to replace a gene, $C_{l,}$ randomly selected from the second parent. By viewing genes as sets consisting of lists of record labels, the gene C_l from the second parent can be eliminated from C_k by *subtracting*, in the set theoretic sense, it from the set corresponding to C_k. The remaining record labels in the gene C_l of the second parent, after set subtraction, will be redistributed to its other genes in such a way that genes that are smaller in size, i.e. containing less records, will have greater chance to be assigned the new record labels. This is done mainly to prevent clusters of very small sizes from evolving. To favor smaller size clusters, we have adopted the following approach. It should be noted, however, that there are other ways of favoring relatively small-size clusters in the redistribution process. The actual approach will likely be dependent on the application domain and how much relatively small-size cluster can be tolerated in the application.

To re-distribute records in a chromosome of [[s_1, s_5], [s_2, s_4, s_{10}], [s_3, s_6, s_7, $s_{8,}$ s_9]], for example, since the length of the list is different (i.e., the number of records in each corresponding cluster is different) and since we would prefer to minimize clusters of

smaller sizes, the probability of remaining records being re-inserted into a list is made to be inversely proportional to the length of the list using the following equation:

$$\Pr(C_k) = \frac{1/|C_k|}{\sum_{k=1}^{K}(1/|C_k|)}$$

Hence, the probabilities of insertion of C_1, C_2 and C_3 are 0.484, 0.324 and 0.194 respectively. The assignment of genes will be done by a mechanism called the roulette wheel selection [Falkenauer 1998]. The probabilities of the genes are added to obtain the corresponding cumulative probabilities.

The cumulative probabilities of C_1, C_2 and C_3 are 0.484, 0.808 and 1 respectively.

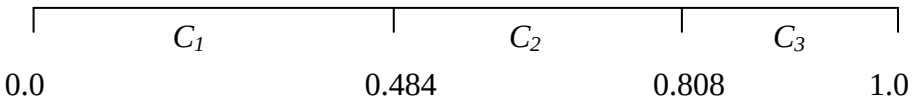


Figure 3-8. Roulette wheel selection of gene for insertion

A random number between 0 and 1 is generated. If it falls between any two numbers, the gene corresponding to that part will be selected for the insertion. The insertion repeats until all remaining records are assigned.

An example of the crossover process is given below. Assume that we have two chromosomes with their list-of-list representation as follows:

Parent$_1$: [[s_1, s_4, s_5], [s_2, s_3, s_7], [s_6, s_8]]

Parent$_2$: [[s_3, s_7], [s_1, s_2, s_4, s_5], [s_6, s_8]]

Each chromosome consists of three lists representing three clusters. Figure 3-9 depicts an example of such reassignment.

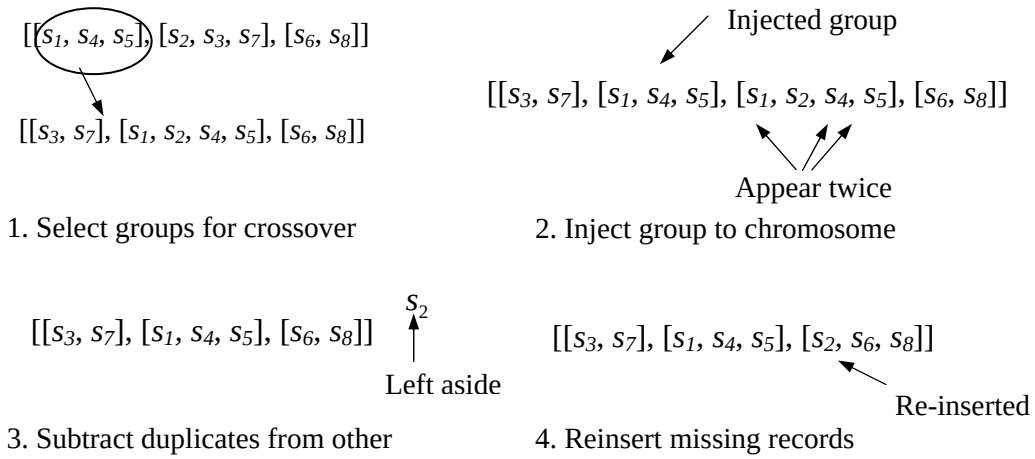


Figure 3-9. List-of-list crossovers

After re-insertion, the replaced list may have overlapped with one or more other lists in the same chromosome. So, duplicated records will have to be removed from the lists. If the removal of records causes some lists to become empty, then the cluster is considered to have disappeared. Then, the crossover operation will be considered failed because it does not contain enough *K* clusters. If this is the case, the crossover operation will be repeated. The crossover operation, it should be noted, will also be repeated when the children produced has the same or a smaller fitness value than the parents due to the steady-state reproduction scheme the GACDD adopts. This is done, as previously discussed before, to prevent premature convergence.

```
1   func crossover(←Pt[i], ←Pt[j], →list3, →list4)
2       index1 = rand(1…| Pt[i]|); index2 = rand(1…| Pt[j]|);
3       /* Swap the lists between the two chromosomes */
4       list3 = merge(Pt[i], Pt[j][index2], index1);
5       list4 = merge(Pt[j], Pt[i][index1], index2);
6       return list1, list2;
7
8   func merge(←list2, ←list1[index2], ←index1)
9       add_list(list1[index2], list2)        /* Add the list to another list-of-list */
10      /* For each record in the index1th list2 */
11      for i =1 to |list2[index1]|
12          /* For each record in the index2th list1 */
13          for j =1 to | list1[index2]| if (list2[index1][i]== list1[index2][j]) do
    delete(list2[index1][i]); od fi
14      for i =1 to |list2|
15          roulettewheelreplace(list2, list2[index1]); return chrom;
16
17  proc roulettewheelreplace(↔list2, ←list2[index1])
```

```
18      for i = 1 to |list2|
19          prob[i] =1/|list2 [i]|
20      prob[i] = prob[i]/sum(prob[])
21      for (i = 1 to |prob[]|) prob_total= prob_total + prob[i];  prob[i] =
    prob_total;
22      for j = |list2[index1]| do
23          k = rand(0…1);
24          for (i = 1 to |prob[]|) if k <  prob[i] do add(list2[i], record) od fi
25      od
```

Figure 3-10. Pseudo code for Crossover

3.4.5. Mutation

To introduce variations into a population, GACDD makes use of two kinds of mutation operators. The first kind of mutation operator selects a cluster randomly. A record label is then randomly selected from this cluster. This label is then randomly placed into another cluster. This operator can be called a relocation operator. The number of record labels moved from one cluster to another is proportional to a mutation rate. This kind of mutation may evolve clusters of very different sizes if the relocation often happens in only few clusters.

The second kind of mutation operator involves randomly selecting two different clusters. A record label from each cluster is then randomly selected and then swapped. This second mutation operator can ensure that the clusters involves have the same size as there will not any change in their sizes.

The mutation rate decides how often mutation takes place. The number of mutation *nMutate* is computed as $MR * N$ where MR is mutation rate and N is number of records in the dataset.

It should be noted that the mutation rate does not need to be held constant. In the initial iterations of the evolutionary process, the number of mutations is set relatively large as this can increase diversity among chromosomes. As evolution progresses, the number of mutation can be stepwise decreased so as to prevent good characteristics of chromosomes from being destroyed.

```
1  proc mutate(↔list)
2      index1 = rand(1… |list|); index2 = rand(1…|list|)
3      mutate_index = rand(1…|list[index1]|)
4      add_record(list[index2], list[index1][mutate_index])
5      delete_record(list[index1][mutate_index])
```

Figure 3-11. Pseudo code for Mutation

3.4.6. Reclassification

GA stochastically finds the solution. It relies on crossover keeping the best genes in the chromosome while bases on mutation looking for better genes for a better solution. In order to get a better solution, mutation needs to pick up the right gene in the right position of the chromosome. At the beginning of the generation, mutation can improve rapidly the quality of chromosome because many genes are not in their best values. Evolution keeps the improved chromosome to the next iteration. After several iterations, there left only some genes with wrong values in the chromosome. At this time, it is difficult for mutation operator to find exactly the right position and generate exactly the right value for that position in a totally random nature. It usually takes a very long time to evolve to a better chromosome. For this reason, GA can only get the near-global optimal solution very often after termination. The one to two percent error hardly gets the right correction or it may take very long time to do so.

The problem of the convergence to the global solution is difficult to solve by GA alone, we adopt another mechanism called reclassification. We generate a set of interesting patterns from the chromosome after mutation. The interesting patterns specify under what attribute-values the record is classified into the clustering label. The details of interesting pattern generation will be explained in Section 3.5. The interesting pattern is presented like this:

If s_q *contains* a_{jk}, then s_q *belongs to* C_p with *Weight of Evidence* = W.

For each record processing a specific set of attribute-values, according to the interesting patterns, we can calculate the total weight of the evidence for that record processing in each clustering label. That is, how much confidence we believe the

record should belong to the cluster. Sometimes, the record may show evidences that it belongs to both C_p and C_k when C_p and C_k also show positive values in weight of evidence. For a crisp clustering, we only assign the record to one cluster. Then we will assign the record to the cluster with maximum weight of evidence. For a data set containing 100,000 records, if only 1 record is in the wrong cluster. Evolutionary operators like crossover and mutation usually hardly show their power in such case and take a long time to find the wrong clustering record. However, we can reclassify the records according to the patterns embedded in the chromosome. By summing up the weight of evidence that match an interesting pattern, we can reclassify the records.

After reclassification, we obtain a new clustering of the records. The new clustering can be the same as or different from the clustering before. If the new clustering is different, we then examine whether there is an improvement in the fitness of the chromosome. If fitness value is increased, then we can use the new clustering forming a new chromosome. We then put the new chromosome into reclassification again. This process repeats until there is no improvement. Then, we can generate a final reclassified chromosome. Reclassification according to the interesting pattern can put the incorrectly clustered records back into the right cluster.

```
1   proc reclassify(←list)
2       if Rule.attribute = list[k][q].a_jk
3           newlist = add(s_q, list, Rule.cluster)
4           fitness-function(↔ list)
5       if (get_fitness(newlist) > get_fitness(list) then
6               reclassify(newlist);
7       fi
8
9   proc add(←s, ↔list,←k)
10      add_record(list[k], s)
```

Figure 3-12. Pseudo code for Reclassification

3.4.7. Reproduction and Deletion

During reproduction, in the case of the simple GA, the old generation is usually replaced completely with the new. For the GACDD, however, we choose to adopt

the steady state without duplicate reproduction scheme. In other words, whenever two children are obtained as a result of the crossover of two parents, their fitness values are computed immediately and they are inserted into the existing population only when their fitness values are greater than the fitness value of the least-fit chromosomes. In such case, the least-fit chromosomes are deleted. In [Davis 1991], it is shown that the SSGA without duplicate scheme can effectively increase the diversity of solutions as encoded in a population thereby increasing the chance for the best solution to be evolved during crossover and mutation. Figure 3-13 depicts the details of the SSGA for partitioning problem.

```
1  proc steady-state-replacement(←chrom-array, ↔Pt)
2      /* Sort the chromosomes by the fitness ascendingly */
3      sort(Pt[], ascending);
4      for i = 1 to |chrom-array| replace(Pt[i], chrom-array[i]);
```

Figure 3-13. Pseudo code for Reproduction

3.4.8. Termination of evolutionary process

Evolutionary process terminates after reaching a stopping condition. Termination conditions are the criteria by which a GA decides whether to continue searching or stop the search. Once a set of termination conditions is defined, each of these termination conditions is checked after each generation to see if it is time to stop.

In general, for GACDD, when considering whether or not to continue an evolutionary process, we consider (i) generation numbers, (ii) fitness thresholds, (iii) fitness convergence and (iv) population convergence. For (i), it means that a maximum number of iteration is defined. GACDD stops execution when the maximum number of generation is reached. For (ii), the fitness threshold, if reached, can terminate the execution of the evolutionary process when the fitness of the best chromosome is greater than or less than a user-defined threshold. For (iii), fitness convergence is concerned with improvement in the fitness values of the best chromosome in a user-defined number of consecutive generations. The fitness of the best chromosome can be said to be converged and the evolutionary process terminated if there is no significant improvement in fitness value. For (iv), the

population convergence is a condition when the variance of the fitness of all the chromosomes in a population is below a user-defined threshold. As this mean that it is very hard to get better solution, it is time to terminate the execution.

In GACDD, we apply the maximum generation termination condition and fitness convergence because they are computational easy. We will explain the purpose of the two termination criteria later. In particular, we describe here how we use the termination criteria involving generation number and fitness convergence.

The generation number condition is set to prevent endless processing. We set the maximum number of generation to be 100,000. Since the population may converge before reaching maximum generations, we also look at the fitness of the best chromosome when deciding whether or not to terminate the evolutionary process. If there is no improvement of the fitness of the best chromosome after 500 consecutive generations, we assume GA has found the best solution and we terminate the evolutionary process. Actually, the maximum number of generation takes little effect in most termination. It just provides a safeguard from endless processing. Almost all trails will end before the maximum number of generation because it is stopped by the 500 consecutive generations.

3.5. The Fitness Function

The choice of the fitness function for a GA can have significant impact on its performance. In general, the fitness functions are different for different problem domains as the characteristics of the best solutions are different from problems to problems.

In general, in finding a fitness function for a genetic algorithm, the function needs to have a well-defined value for all possible solutions and its value should indicate the quality of the solution. For the clustering problem we are considering, for example, the fitness function we are searching for should reflect how good a particular

grouping is. Its value should aim at reflecting how similar the members within a group.

Here, we propose a novel fitness function that can be used to measure the goodness of the current clustering. It is based on measuring the information entropy supporting the records belonging to one cluster to be grouped into the cluster and this measure is called the *weight of evidence* measure. If a record has a positive value of weight of evidence, it means there are evidences supporting it to be grouped into the cluster based on its attributes. In this section, we define the *weight of evidence* measure and how it can be used as a fitness measure for our GACDD. We will show how the grouping of records can be improved through the application of such fitness measure.

3.5.1 A Weight of Evidence Measure

In order to evaluate each chromosome for how good the cluster assignment is, we adopt a measure proposed in [Chan and Wong 1990]. This measure is used here as an objective measure to decide if an attribute value is useful in the characterization of a cluster. If it is useful, the attribute value can be considered as providing evidence supporting a record to belong to the cluster and the amount of evidence can be measured by a *weight of evidence* measure. Given all the attribute values of a record, each of them can then be determined to see if they are useful for the characterization of a particular cluster. If so, there is evidence supporting the record to be in the cluster. The total amount of evidence provided by each useful attribute values can then be used to define as the fitness function of GACDD so that the greater the evidence, the better the grouping arrangement as encoded in a chromosome is.

The fitness value of a chromosome can be determined in two simple steps: 1) finding useful attribute values that provide evidence supporting or refuting a record to be in a particular cluster; 2) compute the total amount of such evidence to determine the fitness values of a cluster grouping encoded in a chromosome. Figure 3-14 shows the pseudo code of the algorithm. In the following, we will discuss each step in detail.

```
1  proc fitness-function(↔ P_t)
2      /* Sort the chromosomes by the fitness ascendingly */
```

```
3
4       for i = 1 to |a|                /* |a| is the number attributes */
5           for j = 1 to |a[i]|         /* |a[i]| is the number of possible values that
        attribute i contains */
6               for k = 1 to |C|        /* |C| is the number of clusters */
7                   findrules(Rule)         /* find the interesting rules */
8
9       for k = 1 to |list|
10          for q = 1 to |list[k]|          /* |list[k]| is the number of records in
        cluster k */
11              for l = 1 to |Rule|
12                  if Rule.cluster = k && Rule.attribute = list[k][q].a_jk
13                          do woe = calculateWOE()
14                                  totalwoe = totalwoe + woe
15                          od
16                  fi
```

Figure 3-14. Interesting patterns + Weight of Evidence

Step 1. Discover useful attribute values

The major goal of this step is to determine if an attribute value, a_{jk}, is useful in determining if a record should or should not be in a particular cluster, C_p. For this purpose, we use a statistics proposed in [Chan *et al.* 1988; Chan and Wong 1990].

If the attribute value a_{jk} is useful, we represent this as $a_{jk} \Rightarrow C_p$. We can define a *support* measure of C_p as below:

$$\text{support}(C_p) = \frac{\text{no. of records in } C_p}{M} = \Pr(C_p) \qquad (1)$$

i.e., it is the probability of C_p in the database and is defined as the number of records containing cluster label C_p, divided by the total number of records.

In addition, we can also define the confidence of $a_{jk} \Rightarrow C_p$ as the conditional probability of C_p given a_{jk}, i.e., it is defined as the number of records containing both a_{jk} and C_p, divided by the total number of records containing, a_{jk} , as follows:

$$\text{confidence}(a_{jk} \Rightarrow C_p) = \frac{\text{no. of records in } C_p \text{ that is characterized by } a_{jk}}{\text{no. of records characterized by } a_{jk}} = \Pr(C_p \mid a_{jk}) \qquad (2)$$

where *M* is the total number of counted records. The value of *M* should be equal to or less than *N* due to the possibility of having missing values in the database.

To know if an attribute value, say a_{jk}, is useful in determining if a record should or should not be in a particular cluster, C_p, we determine if

$$\Pr(\text{records is in } C_p \mid a_{jk}) = \frac{\text{no. of records in } C_p \text{ that is characterized by } a_{jk}}{\text{no. of records characterized by } a_{jk}}$$

is significantly different from $\Pr(\text{records is in } C_p) = \frac{\text{no. of records in } C_p}{\text{no. of records in data set}}$ (3)

This means that we need to decide if support(C_p) is significantly different from confidence($a_{jk} \Rightarrow C_p$). If this is the case, then a_{jk} can be considered as useful.

To determine how great the difference between Pr(records is in $C_p \mid a_{jk}$) and Pr(records is in C_p) (i.e., also as support(C_p) and confidence($a_{jk} \Rightarrow C_p$)} to be considered significant, we use an objective measure based on statistic confidence. Let $o_{c_p a_{jk}}$ be the total number of records in cluster C_p that is also characterized by a_{jk}, and $e_{c_p a_{jk}}$ be the expected number of records in C_p that is also characterized by a_{jk}. Then, $o_{c_p a_{jk}}$ = observed number of records is in $C_p \mid a_{jk}$ and $e_{c_p a_{jk}}$ = observed number of records is in C_p * observed number of records with a_{jk}. Since the absolute difference $|e_{c_p a_{jk}} - o_{c_p a_{jk}}|$ of the expected and observed difference does not reveal relative degree of discrepancy between the two values, an adjusted residual is proposed in [Chan *et al.* 1988; Chan and Wong 1990] to objectively determine if the differences are significantly different and this adjusted residual is defined as:

$$d_{c_p a_{jk}} = z_{c_p a_{jk}} / \sqrt{v_{c_p a_{jk}}} \quad (4)$$

where $z_{c_p a_{jk}} = (o_{c_p a_{jk}} - e_{c_p a_{jk}}) / \sqrt{e_{c_p a_{jk}}}$ (5)

and $v_{c_p a_{jk}} = (1 - \frac{o_{c_p +}}{M})(1 - \frac{o_{+ a_{jk}}}{M})$ (6)

There are two components in the adjusted residual. The first component $z_{c_p a_{jk}}$ is the standardized residual [Chan *et al.* 1988; Chan and Wong 1990] (Equation (5)). It is used to eliminate the influence of marginal totals on the absolute difference. $z_{c_p a_{jk}}$ has the property that it is distributed asymptotically as chi-square and has an approximate normal distribution with a mean of approximate 0 and a variance of approximately 1. Since it is based on the asymptotic variance of 1, when it differs too much from 1, a better estimation is needed to reflect the residual. Hence, it requires the second component.

The second component $v_{c_p a_{jk}}$ (Equation (6)) is the maximum likelihood of its asymptotic variance. It takes into account of two parameters $o_{+a_{jk}}$ and $o_{c_p +}$ which are the number of records having attribute-value a_{jk} and the number of records in C_p respectively. $v_{c_p a_{jk}}$ can reflect more the true distribution of records. The adjusted residual $d_{c_p a_{jk}}$ is obtained from dividing the standardized residual with the new variance $v_{c_p a_{jk}}$. So it can give a better approximation to the case.

In order to know whether a_{jk} is useful in supporting or refuting a record as a member of a cluster C_p, we can make use of the probabilities of the observed and expected occurrence. Since the adjusted residual exhibits an approximate standard normal distribution, if $|d_{c_p a_{jk}}| > 1.96$, we can conclude, with 95 percent (or 2.576 with 99 percent) that the relationship between C_p and a_{jk} is significant. Such relationship can be two ways; it can be due to the presence or the absence of a_{jk}. In other words, a_{jk} is a determinant factor in the assignment of the record to C_p. Three interesting patterns can be obtained by the adjusted residual:

1) Positive Pattern. If $d_{c_p a_{jk}} > 1.96$, we can conclude that with 95 percent confidence a record characterized by a_{jk} is likely to be a member of C_p.
2) Negative Pattern. If $d_{c_p a_{jk}} < -1.96$, we can conclude that with 95 percent confidence a record not characterized by a_{jk} is likely to be a member of C_p.

3) Neutral Pattern. If -1.96 > $d_{c_p a_{jk}}$ > 1.96, we can conclude that the pattern is not significant. A record characterized by a_{jk} does not provide us with much information about whether it should be a member in C_p

By scanning through each attribute, we can obtain 1) the frequency of each attribute value, 2) the frequency of each cluster label and 3) the frequency of the presence of both attribute value and cluster label. After that, we can find out which attribute value is related to the cluster label. Those attribute values with statistically significant adjusted residual are considered useful. According to different level of statistical confidence, when adjusted residual > 1.96, we can conclude that the attribute value involved makes positive evidence supporting a record to be a member of a cluster and if adjusted residual < -1.96, we can conclude that the attribute value involved makes negative evidence refuting a record to be a member of a cluster. With these attribute values identified, we can determine which if a record should be in a particular cluster by determining if it possesses useful attributes. To do so, we still require a quantitative measurement to determine the degree of evidence for such assignment and the details of this process are given below.

2) Compute Weight of Evidence Measures

The attribute values that are useful for the characterization of clusters identified can provide varying amount of evidence supporting and refuting a record to be assigned to a cluster. There is a chance that a record may be characterized by some attribute values supporting it to be a member of a group and at the same time, it also contains attribute values refuting it to be a member of that group. Similarly, there is a chance that some of the attribute values of a record support it to be a member of a cluster and some support it to be another. Given these considerations, there is a need for us to measure the amount of evidence supporting or refuting the cluster membership of a cluster given the attribute values characterizing a record. Here, we propose a *weight of evidence* measure, W if a record is characterized by a_{jk} then that a record belongs to C_p is with weight of evidence W (record in C_p / record not in C_p | record characterized by a_{jk}) where:

$$W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jk}) = I(\text{Cluster} = C_p \mid a_{jk}) - I(\text{Cluster} \neq C_p \mid a_{jk})$$

where
$$I(\text{Cluster} = C_p : a_{jk}) = \log \frac{\Pr(\text{Cluster} = C_p \mid a_{jk})}{\Pr(\text{Cluster} = C_p)} \quad (7)$$

$$I(\text{Cluster} \neq C_p : a_{jk}) = \log \frac{\Pr(\text{Cluster} \neq C_p \mid a_{jk})}{\Pr(\text{Cluster} \neq C_p)} \quad (8)$$

which can be further defined as follow:

$$I(\text{Cluster} = C_p : a_{jk}) = \log \frac{\text{confidence}(a_{jk} \Rightarrow C_p)}{\text{support}(C_p)} \quad (9)$$

$$I(\text{Cluster} \neq C_p : a_{jk}) = \log \frac{\text{confidence}(a_{jk} \Rightarrow \overline{C_p})}{\text{support}(\overline{C_p})} \quad (10)$$

Weight of evidence is a measure of the amount of positive and negative evidence supporting and refuting a record to be clustered into C_p. It is based on the concept of mutual information [Cover and Thomas 1990; Lubbe 1997].

$$I(\text{Cluster} = C_p : a_{jk}) = \log \frac{\Pr(\text{Cluster} = C_p \mid a_{jk})}{\Pr(\text{Cluster} = C_p)} \quad (11)$$

$I(\text{Cluster} = C_p : a_{jk})$ is positive if and only if $\Pr(\text{Cluster} = C_p \mid a_{jk}) > \Pr(\text{Cluster} = C_p)$; otherwise, it is negative or zero. Mutual information measures the decrease (if positive) or increase (if negative) in uncertainty about when record assign to the cluster C_p given the attribute-value a_{jk}. The difference in the gain in information between records assigned to the cluster C_p, records assigned not to C_p given attribute-value a_{jk} is a measure of evidence that a_{jk} favors the assignment the record to cluster C_p as opposed to other cluster not in C_p. The difference, denoted by W (record in C_p / record not in C_p | record characterized by a_{jk}), is defined as weight of evidence:

$$W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jk}) = \log \frac{\Pr(a_{jk} \mid \text{Cluster} = C_p)}{\Pr(a_{jk} \mid \text{Cluster} \neq C_p)} \quad (12)$$

$$= \log \frac{\Pr(a_{jk} \cap \text{Cluster} = C_p)}{\Pr(a_{jk} \mid \text{Cluster} \neq C_p)} \quad (13)$$

Since we will not allow empty cluster, during initialization, crossover and mutation, Pr (C_p) where p = 1, …, n cannot be 0. That implies Pr ($\overline{C_p}$) also cannot be 0. However, we should note that there exist cases where there are no records containing

both C_p and a_{jk}, so Pr ($a_{jk} \cap C_p$) is 0. The same applies for Pr ($a_{jk} \cap \overline{C_p}$). If this is the case, then weight of evidence will be 0 and undefined respectively. So, we must replace the values with some more meaningful. If Pr ($a_{jk} \cap$ not $\overline{C_p}$) = 0, weight of evidence $W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jk})$ is replaced by $\max_{k,l} W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jl})$ given Pr ($a_{jk} \cap \overline{C_p} \neq 0$, where l = 1, 2, …, L and L is the total number of all possible values of A_j. Similarly, if Pr ($a_{jk} \cap C_p$) = 0, weight of evidence $W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jk})$ is replaced by $\min_{k,l} W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jl})$ given Pr ($a_{jk} \cap C_p$) ≠ 0, where m = 1, 2, …, M and M is the total number of all possible values of A_j. Given all attribute-values, we just calculate weight of evidence for those patterns that is significant. We then make use these weights of evidence to calculate the fitness values of each chromosome.

3) Determine Fitness Value

Given a data set and a chromosome, a data set S is a set of records $s_q \in S$, q = 1, …, N where N is the total number of records. A record s_q corresponds to a row of such data set and an attribute, A_j, j = 1, …, n where n is the number of attributes, corresponds to a column of each record. a_{jk} is the value of the attribute A_j in the record s_q. For each record s_q, we determine, for each cluster encoded in a chromosome, how much evidence its attribute values provide supporting or refuting the record to be a member of the cluster. For each attributes useful in determining the membership of the record of a particular cluster, we add up the weight of confidence provided by these attributes. By repeating this for each possible cluster encoded in a chromosome, the amount of evidence supporting and refuting s_q to belong to C_p, can easily be determined.

Suppose a record s_q is characterized by m attribute values and only some of them are found to be useful in determining if s_q should be assigned to C_p, then we can define a total weight of evidence, provided by all useful attribute values, that the record belonging to C_p will be equal to the sum of weight of evidence, i.e.,

$$W_{s_q c_p} = \sum_{j=1}^{m} W(\text{Cluster} = C_p / \text{Cluster} \neq C_p \mid a_{jk}) \qquad (16)$$

We can make use of this value to determine the fitness of a chromosome with a particular cluster arrangement encoded in it.

The total fitness of a chromosome can be obtained through the total weight of evidence of individual cluster, i.e.,

$$fitness = \sum_{p=1}^{P} \frac{\sum_{q=1}^{N} W_{s_q c_p}}{no.\ of\ records\ in\ C_p} \qquad (17)$$

The fitness of a chromosome is the summation of the average weight of evidence of all cluster labels. As the weight of evidence implies the amount of information or evidence that support or refute a record to be assigned to a particular cluster, greater evidence implies greater probability the record will be in the cluster. So, we try to assign a record if it maximizes the evidence. In other words, the greater the evidence, the greater the probability that the record is correctly assigned. After we summed up the evidence of all records; the greater the total evidence, the greater in general will be each record in its best assignment, hence, the chromosome will likely be fitter.

Chapter 4

Experimental results

In this chapter, we present the results of our attempts to evaluate GACDD. To do so, we used both synthetic and real data and compared the performance of the proposed clustering algorithm against several different popular clustering algorithms as described in Chapter 2: (1) the *K*-means [MacQueen 1967] algorithm which is perhaps the most popular clustering algorithm amongst all; (2) the *K*-modes [Huang 1997a, 1997b, 1998] algorithm which is a clustering algorithm developed for use with categorical data; (3) a GA based clustering algorithm [Cristofor and Simovici 2002; Simovici *et al.* 2000; and Simovici *et al.* 2002]; (4) the COBWEB [Fisher 1987] which is a machine learning approach that can be used to cluster discrete data; and (5) Autoclass [Cheeseman *et al.* 1988; and Cheeseman and Stutz 1995] which is another popular machine learning method that is parametric and that makes use of the idea of Bayesian classification when performing clustering.

In the following sections, we first describe the data sets we used in our experiments, and then we discuss the criteria we used in our performance evaluation.

4.1. Data Sets and Evaluation Criteria

To evaluate the effectiveness of the proposed clustering algorithm, we test it with both the synthetic and real data. In this section, we explain the data sets we used and the experiments we carried out.

4.1.1. Data sets

The data sets we used in our experiments contain both synthetic and real data. The synthetic data set is used as a controlled data set to study the performance of the proposed algorithm with respect to their ability to handle missing and noisy data. Besides synthetic data, we also used real data sets to study how well the proposed clustering algorithm can handle real data. These real data sets were selected from a wide variety of application domains including agricultural, medical and social databases. The set size ranges from several hundreds to several tens of thousand records. Among these, most of them are data sets containing all categorical attributes. For those data sets containing numeric attributes, equal-width discretization is performed ahead of time. Missing values, it should be noted, can be found in varying percentages ranging from 0% to 50% in these data sets.

4.1.2. Experimental Set-up

For performance benchmarking, we used the *K*-means, *K*-modes and COBWEB algorithm implemented in WEKA [Witten and Frank 1999]. We also used AutoClass implemented in [Cheeseman and Stutz 1995]. For the GA-based clustering algorithm, we implemented the algorithm according to [Cristofor and Simovici 2002].

For each experiment, we performed 1000 trials with each data set using different parameters. For each trial, the data in the data set is shuffled to prevent the ordering effect on the algorithm such as COBWEB. The averages of all results obtained from all different trials were then computed and recorded.

All algorithms, except COBWEB, require the number of clusters to be pre-defined. For the purpose of comparison of accuracy of the various algorithms, we merged clusters to get the desired number of clusters for COWEB. The standard deviation was also recorded.

The set-up for our experiments with each of the different algorithms is described as follows. For GACDD, we set the crossover and mutation rate to 0.1 and 0.01 respectively. As discussed in Chapter 3, we increase the mutation rate and decrease

the crossover rate during the evolutionary process. For all experiments, the population size was set to 100 and the maximum number of reproductive iterations was set to 10000.

For the *K*-means and *K*-modes algorithms, the choice of initial centers may affect the results of clustering. To find these centers, a parameter called random seed is used for the initial cluster-center selection. We used another random number generator to generate random seeds for each trial.

For COBWEB as implemented in WEKA, we need to supply two parameters. One is the *acuity* for the standard deviation that is only effective for numeric attributes. Another is the *cutoff*, which determine the split and merge conditions. The cutoff value is between 0 and 1 and is set to allow us to determine the number of clusters in our experiments. We increase the cutoff value until the correct number of cluster is found.

For the GA-based clustering algorithm, it does not require any parameter to be set. For Autoclass, the statistical model needed to be specified for it to work. Since the attributes are categorical in nature, we select the binomial model. The maximum number of iterations for Autoclass was set to 10000 in our experiments

When comparing performance of the various clustering algorithms, we chose to use the misclassification counts as defined in [Krovi 1992; Li and Biswas 2002; and Huang 1997b] as a measure of accuracy. If a clustering algorithm is effective, the clusters it discovers should be of good quality and should share common patterns. A good classifier-discovery algorithm should be able to uncover these patterns and use them to correctly re-classify records randomly selected from each cluster. The misclassification count can, therefore, be used to determine how good the clusters formed are.

For our case, how the data records should be grouped to form clusters is actually known. We can therefore easily determine the misclassification count and be able to

use it to evaluate the relative merit of all clustering algorithms. The misclassification count is a function of an accuracy count that can be defined as:

$$accuracy = \frac{\text{total number of records in database - number of records misclassified}}{\text{total number of records in database}}$$

Since some clustering algorithms such as COBWEB does not allow pre-setting of the number of clusters at the beginning of each experiment, for the purpose of comparing results, we adopted the following procedures: (i) if the number of clusters discovered by COBWEB is too small, the clusters are mapped against the original grouping in such a way so as to obtain the highest accuracy; (ii) if the number of clusters discovered is too large, then do (i) to achieve the highest possible accuracy for those discovered clusters and for those records in the remaining clusters, they are assigned to existing clusters in such a way that the accuracy is maximized. By doing so, we can obtain the highest possible accuracy with COBWEB even when the number of clusters it generates is different from the desired solution.

Since we have pre-defined the number of clusters, some records must be correct no matter how they are grouped and this is called the *baseline accuracy* here. For example, consider 100 records in a database containing 4 clusters with 25 each, the baseline accuracy will be 25%, i.e., if we are to group every record into one single cluster, we have an accuracy of at least 25%. For this reason, the accuracy of different algorithms should be benchmarked also against the baseline accuracy instead of considering only its absolute value.

4.2. Results

4.2.1. The synthetic Database

The synthetic dataset contains 100 houses divided into 4 clusters. To embed patterns in these clusters, we define a set of if-then rules that specify what values some of the attributes should take.

The dataset is characterized by 9 attributes that are categorical in nature (see Table 4-1). In order to test the clustering algorithms' ability to handle noisy and missing data, the data values were randomly selected and removed or corrupted by replacing them with other values.

Attributes	Values
(1) Length of Funnel	{Short, Long}
(2) Number of Funnels	{1, 2, 3, 4}
(3) Shape of Window	{Square, Circle}
(4) Position of Window	{Left, Middle, Right}
(5) Style of Window	{Cross lined, Horizontal lined, Vertical lined}
(6) Texture of Door	{Black, Horizontal lined, Vertical lined, Checked}
(7) Position of Door	{Left, Middle, Right}
(8) Position of Plant	{None, Left, Right, Both}
(9) Texture of Roof	{None, Blacked, Checked, Horizontal lined, Vertical lined}

Table 4-1. Attributes in the synthetic dataset.

When generating the data, we used the following rules. It should be noted that two attributes (3) Shape of Window and (5) Style of Window are irrelevant for the characterization of any cluster but are included mainly to test the ability of the clustering algorithm to handle irrelevant data. The attribute-values of these two attributes are randomly generated. For a clustering algorithm to perform well, it has to be able to ignore these two attributes during the cluster formation process. An effective cluster should be able to discover the patterns as defined by these rules below.

Class 1:
1) (2) Number of Funnels = 3
2) (2) Number of Funnels = 4
3) (7) Position of Door = Middle
4) (7) Position of Door = Right

Class 2:
5) (8) Position of Plant = Left.
6) (1) Length of Funnel = Long.
7) (6) Texture of Door = Horizontal lined.

Class 3:
8) (2) Number of Funnels = 2
9) (4) Position of Window = left

10) (9) Texture of Roof = Horizontal lined
11) (1) Length of Funnel = Short

Class 4
12) (6) Texture of Door = Black
13) (6) Texture of Door = Checked
14) (1) Length of Funnel = Short
15) (8) Position of Plant = Both
16) (9) Texture of Roof = Vertical lined

Figure 4-1. Simulated data of houses

In order to make *K*-means work with categorical attributes, the categorical attributes have to first be made into binary attributes. For example, a categorical attribute "color" carries three possible values "red", "blue" and "yellow". The "color" attribute will be required to split into the attributes of "red color", "blue color" and "yellow color". If the data record originally carries "color" with "red" value, it will make the "red color" one and the other two colors zero. Hence, the total number of attributes in our synthetic data set is increased from 9 to 30. Also, for *K*-means or *K*-modes, when handling missing values, they are replaced, in our experiments, by the mode of an attribute.

The results of the clustering experiments are shown in the Table 4-2. As shown in the table, our proposed algorithm performed better than *K*-means and *K*-modes. Our method can classify the records with no error all the time. The performance of *K*-means is somehow poor. It may be due to the reason that it needs to transform the data into a Boolean value. Such distance measure is not appropriate for categorical attributes. Also, *K*-means does not have any mechanism to jump from a local optimum to other searching space to reach a global maximum. So its performance is subject to the initial centers selected. If these are not chosen well, it may have poor result. For this reason, the results obtained cannot be guaranteed to have good quality. In fact, there can be a large discrepancy among different results.

Compared to *K*-means, by the schemata theory [Michalewicz 1996] and many empirical studies [Davis 1991; Goldberg 1989; and Michalewicz 1996], GA usually can achieve good results. From our experiments, this proves to be the case as we have 100% accuracy in all our 1000 trials.

In the cases of testing the proposed algorithm, we also test with real data sets, since real data often contains many missing and inconsistent, erroneous values, it is important that a clustering algorithm be capable of handling these data. From our experimental results, our algorithm can be very effective as it achieves high accuracy with such data.

Other than being able to handle noisy data, it is important for a clustering algorithm to discover the more important attributes that are responsible for characterizing or discriminating one cluster from another. There are two advantages if these attributes can be identified. First, the dimensionality problem can be better handled. Second, we can better interpret the clustering results. For the proposed method, we are able to also explicitly discover which attribute values are responsible for characterizing a particular cluster.

Without Missing Value and Noise

When the dataset contains no missing values or noise, the results are given as follows.

	Accuracy	Rank
GACDD	100.00%	1
K-means	66.00%	6
K-modes	72.80%	5
COBWEB	100.00%	1
GA Clustering	93%	4
Autoclass	100.00%	1

Table 4-2. Accuracy for the simulated data without missing value and noise.

As shown in Table 4-2, our method performs better than *K*-means, *K*-mode and the GA-based algorithm and has the same accuracy as COBWEB and Autoclass. The performances of the *K*-means and *K*-modes algorithm are relatively poor and the local-optimum problem as discussed earlier may also be a dominant cause for the relatively poor performance of these algorithms. Also, for *K*-means, it does not explicitly handle categorical data. It needs to rely on binarization to handle them. As *K*-means and *K*-modes adopt the same algorithm, the performance difference is mainly due to the dissimilar distance measures and this is the likely reason why *K*-means does not handle categorical data as well as *K*-modes.

It should be noted that COBWEB discovered 3 clusters, which are of sizes 25, 25 and 50 respectively. Looking at the second level, the large cluster can be split into two clusters of size 25 each. As a result, although COBWEB also matches all four classes

and contains 100 percent accuracy, it requires further processing to obtain the required number of clusters. It usually is not easy for normal user.

With 10% Missing Values and Noise

To investigate into the effectiveness of the different clustering algorithms in the presence of difference missing values and noise level, we randomly added 10% noise to the original dataset. For the proposed algorithm, COBWEB, GA-based clustering and Autoclass, they have been designed to handle missing values by themselves. For *K*-means and *K*-modes, the missing values in the dataset are replaced by the highest frequency values.

	Accuracy	Rank
GACDD	98.5%	1
K-means	84.3%	5
K-modes	74.5%	6
COBWEB	95.7%	2
GA Clustering	90.2%	4
Autoclass	94.2%	3

Table 4-3. Accuracy for the simulated data with 10% missing value and noise.

Of these different algorithms, GACDD performed better. COBWEB and Autoclass also show satisfactory results. The GA based clustering algorithm also performed quite well and is relatively stable but it does not find the best solution. For *K*-means and *K*-modes, there is a surprisingly improved accuracy. Theoretically, *K*-means and *K*-modes do not have special capability to handle noise and missing value. However, you will notice later that as the noise levels vary, the accuracy of these approaches can also vary significantly.

With 20% and 50% Missing Value and Noise

When missing values and noise are increased to 20%, GACDD still give relatively good results when other algorithms have shown rather significant degrade in performance. Autoclass also performed quite well at this noise level. This is because COBWEB and Autoclass also process certain capability in handling such data. Compared to them, *K*-means and the others are rather far-behind and there is a rather substantial difference in terms of classification accuracy.

	Accuracy	Rank
Our algorithm	88.6%	1
K-means	76.2%	4
K-modes	73.2%	5
COBWEB	80.5%	3
GA Clustering	62.1%	6
Autoclass	86.6%	2

Table 4-4. Accuracy for the simulated data with 20% missing value and noise.

When missing values and noise are increased to 50%, GACDD is still relatively stable in performance while the other algorithms have dropped sharply in performance. Table 4-5 shows the accuracy against the level of noise and missing value.

	Accuracy	Rank
Our algorithm	80.6%	1
K-means	31.2%	5
K-modes	30.7%	6
COBWEB	70.3%	3
GA Clustering	51.8%	4
Autoclass	74.2%	2

Table 4-5. Accuracy for the simulated data with 50% missing value and noise.

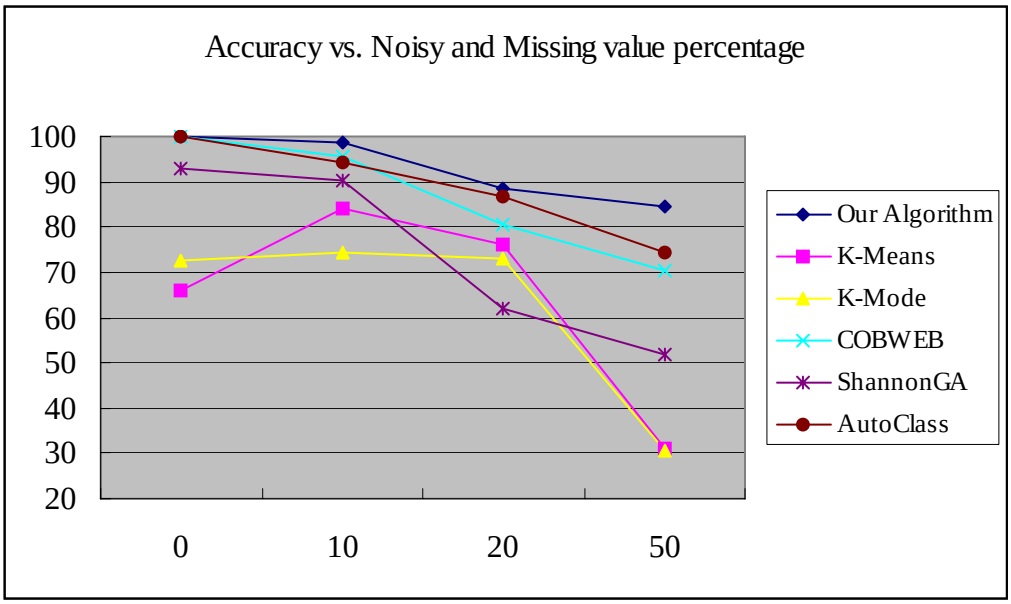


Figure 4-2. Accuracy vs. Noisy and Missing value percentage.

4.2.2. The soybean Database

The soybean database [Blake and Merz 1998] contains 47 records and each record is characterized by 36 attributes. It is one of the most typical data sets used to test clustering algorithms. All attributes in the dataset are categorical in nature. Among the attributes, one attribute is the cluster label which specifies the disease that can be carried by the soybean species. These diseases include: the Diaporthe Stem Canker, the Charcoal Rot, the Rhizoctonia Root Rot and the Phytophthora Ort. The first three classes have 10 records for each and the last class has 17 records. So, the clusters are similar in size.

Records in the Rhizoctonia Root Rot and the Phytophthora Ort share many common features. Since the records with the same disease are placed consecutively together in the data sets, we reshuffle the records to examine the effect of ordering of records that are sensitive to some clustering method such as COBWEB.

In our experiment, since the *K*-means algorithm cannot handle categorical data well, there is a need to transform all categorical attributes in the dataset to binary-valued attributes. After transformation, the number of attributes is, therefore, expanded from 35 attributes to 84 attributes for *K*-means. The clustering result is illustrated in the Table 4-6.

	Accuracy	Rank
GACDD	100%	1
K-means	84.5%	6
K-modes	88.9%	5
COBWEB	100%	1
GA clustering	94.5%	4
Autoclass	100%	1

Table 4-6. Accuracy for the soybean dataset.

From the above result, GACDD performed as well as COBWEB and Autoclass. It should be noted, however, that COBWEB is sensitive to the ordering of the records while others do not. When the records are reshuffled, the conceptual tree formed in COBWEB is different. It happens some results that cannot form enough clusters in a particular tree level. For these cases, we manually split the sub-clusters in the tree

node till enough clusters are formed. In the experiment, we match the solutions against the clusters to obtain the greatest possible matches by merging and splitting different branches to get the desired number of clusters. However, in real life database, the manual grouping is difficult to handle like the selection of cut-off level in hierarchical clustering. We will discuss more on this issue in later section.

The GACDD gets very good results in this data set. The GA based clustering algorithm obtained relatively satisfactory results. It is better than *K*-means and *K*-modes but not as good as GACDD. *K*-means and *K*-modes always get local maximum. Relatively speaking, GA-based algorithms can better tackle this problem.

4.2.3. The zoo Database

The zoo database [Blake and Merz 1998] contains 101 records. Each record is characterized by 18 attributes that describe the features carried by the animals. 15 out of the 18 attributes are binary-valued. The binary attributes are Hair, Feathers, Eggs, Milk, Airborne, Aquatic, Predator, Toothed, Backbone, Breathes, Venomous, Fins, Tails, Domestic, and Catsize. One attribute is numeric that specifies the number of legs that contains 0, 2, 4, 5, 6 and 8. One attribute is the animal name that is irrelevant and it is ignored. One is the class that describes the specimen. We regarded the specimen as the natural clustering and used for result comparison. There are no missing values and there are a total of 7 specimens, of which there are 41 mammals, 20 birds, 13 fishes, 10 invertebrates, 8 insects, 5 reptiles and 4 amphibians. The seven specimens varied greatly in the number of animals. Hence, we can test the different algorithms in handling unequal clusters size. In our experiment, since all the attributes are categorical, there is a need to transform all the attributes to binary-valued attributes for *K*-means algorithm. Therefore, the number of attributes is expanded from the original 16 attributes to 38 binary-valued attributes. The clustering result is tabulated in Table 4-7.

	Accuracy	Rank
GACDD	92.1%	1
K-means	74.7%	6
K-modes	75.7%	5

COBWEB	89.6%	2
GA clustering	78.3%	4
Autoclass	87.1%	3

Table 4-7. Accuracy for the zoo dataset.

As illustrated in Table 4-7, GACDD performed relatively well. It performed well with different cluster sizes. It can correctly identify the largest cluster that has 41 records and the smallest cluster that has only 3 records. Although COBWEB has very high accuracy in the zoo data set, it could not discover the correct number of clusters. The closest numbers of clusters obtained are only 6 and 8. The smallest cluster that describes reptiles with record size four was never found.

For the GA based clustering algorithm, it has an average accuracy of 78.3% but there is a large variance in accuracy in different trials. In fact, the standard deviation is as high as 6.67% as opposed to 1.7% in case of GACDD. The reason for this is that GA clustering has no quick methods but only mutation to correct the wrong clustered records when it hits a local optimal. In contrast, our algorithm can use the patterns discovered in the data to more accurately place clustered records into the correct one.

The accuracy of *K*-means and *K*-modes are the lowest. This is likely due to these algorithms not being able to handle clusters with wide-varying sizes [Han and Kamber 2001].

4.2.4. The vote Database

The vote database [Blake and Merz 1998], which describes the 1984 United States Congressional Voting Records have 435 votes on 16 key issues. There are 17 attributes, of which 16 are Boolean-valued. The voting issues include handicapped-infants, water-project-cost-sharing, adoption-of-the-budget-resolution, physician-fee-freeze, el-salvador-aid, religious-groups-in-schools, anti-satellite-test-ban, aid-to-nicaraguan-contras, mx-missile, immigration, synfuels-corporation-cutback, education-spending, superfund-right-to-sue, crime, duty-free-exports, export-administration-act-south-africa.

Each record is characterized by a classification label describing that the vote is coming from democrat or republican. There are 267 democrats and 168 republicans. All attributes contain 2% to 25% missing values with an average of about 12% missing. Since all attributes are binary in nature, there is no need for any transformation to be done for *K*-means. The clustering result is illustrated in Table 4-8.

	Accuracy	Rank
GACDD	89.3%	1
K-means	83.6%	6
K-modes	86.4%	4
COBWEB	88.0%	2
GA clustering	86.4%	4
Autoclass	87.3%	2

Table 4-8. Accuracy for the vote dataset.

From the experiment, the accuracy of all the algorithms is relatively close to each other. The differences among them are less than 6%. Nevertheless, GACDD still outperform others. The reason for small differences in accuracy may be due to only 2 clusters are present in the record set. *K*-means is relatively poor amongst all. For most of the trials, *K*-means is relatively good. However, perhaps due to poor initialization of centers, the error rate can go up to 45% in some cases.

4.2.5. The heart Database

The heart database [Blake and Merz 1998] contains 4 sets of data, which include the Cleveland, Hungarian, Switzerland and Long Beach VA data sets. They record the symptoms of patients with heart diseases. The database contains 14 attributes. Among these attributes, 6 are continuous: age, resting blood pressure, serum cholesterol, maximum heart rate achieved, ST depression induced by exercise relative to rest and number of major vessels colored by fluoroscopy. The remaining 8 are categorical attributes. They are: sex, chest pain type, fasting blood sugar, resting electrocardiography results, exercise induced angina, the slope of the peak exercise ST segment, heart test and heart disease presence. The last attribute can take on 0 and

1 which distinguish the absence and presence of the heart disease, so there are two natural clusters.

There are missing values in two attributes which are: the number of major vessels and the chest pain type. The categorical attributes are transformed into binary-valued attributes and expanded into 23 attributes for *K*-means. The numeric attributes are discretized into equal-depth intervals for GACDD, and the GA-based clustering algorithm and *K*-modes. For COBWEB and Autoclass, no data transformation is required. The cluster results are illustrated in Table 4-9.

	Accuracy	Rank
GACDD	80.1%	1
K-means	72.6%	4
K-modes	69.1%	5
COBWEB	76.0%	2
GA clustering	65.7%	6
Autoclass	74.5%	3

Table 4-9. Accuracy for the heart-disease dataset.

The GACDD outperforms the other algorithms in this experiment. It has an average of 80.1% accuracy throughout the experiment. The experimental results show that the proposed algorithm can perform effectively in the presence of both continuous and discrete data simultaneously.

It should be noted that, for GACDD and the GA based clustering algorithm and *K*-means and *K*-modes, we have tried to discretize the continuous intervals into different number of intervals, ranging from 3 to 10. The overall accuracy does not vary greatly and there is just a maximum of 2% difference. For COBWEB, in some of the trials, three clusters were discovered in the first level of the concept hierarchy whereas only there are two clusters in the original database. The accuracy of COBWEB obtained in this case is thus by merging the extra cluster to one of the other clusters using the mechanism described in Section 4.1.2.

Although the performance of *K*-means is better, there is a large difference among the accuracy in trials again. This may be due to a wrong choice of cluster centers in some

cases. For *K*-modes, the accuracy through trials are more stable, however, there its accuracy is relatively low. For the GA based clustering algorithm, it cannot find the clusters underlying the data and this is likely due to the inability of the simple entropy measure to discover good clusters.

4.2.6. The mushroom Database

The mushroom database [Blake and Merz 1998] that contains data about 23 species of gilled mushrooms in the Agaricus and Lepiota Family, contains 8124 records in the data set. Each species is characterized as definitely edible and poisonous by its class. Each record carries 22 attributes that are all nominally valued. These values describe the appearance, population and habitat of all the mushrooms. All attributes are shown in the Table 4-10.

Attribute	Value
(1) Cap-shape	{Bell, Conical, Convex, Flat, Knobbed, Sunken}
(2) Cap-surface	{Fibrous, Grooves, Scaly, Smooth}
(3) Cap-color	{Brown, Buff, Cinnamon, Gray, Green, Pink, Purple, Red, White, Yellow}
(4) Bruises	{Bruises, No}
(5) Odor	{Almond, Anie, Creosote, Fishy, Foul, Musty, None, Pungent, Spicy}
(6) Gill-attachment	{Attached, Descending, Free, Notched}
(7) Gill-spacing	{Close, Crowded, Distant}
(8) Gill-size	{Broad, Narrow}
(9) Gill-color	{Black, Brown, Buff, Chocolate, Gray, Green, Orange, Pink, Purple, Red, White, Yellow}
(10) Stalk-shape	{Enlarging, Tapering}
(11) Stalk-root	{Bulbous, Club, Cup, Equal, Rhizomorphs, Rooted}
(12) Stalk-surface-above-ring	{Ibrous, Scaly, Silky, Smooth}
(13) Stalk-surface-below-ring	{Ibrous, Scaly, Silky, Smooth}
(14) Stalk-color-above-ring	{Brown, Buff, Cinnamon, Gray, Orange, Pink, Red, White, Yellow}
(15) Stalk-color-below-ring	{Brown, Buff, Cinnamon, Gray, Orange, Pink, Red, White, Yellow}
(16) Veil-type	{Partial, Universal}
(17) Veil-color	{Brown, Orange, White, Yellow}
(18) Ring-number	{None, One, Two}
(19) Ring-type	{Cobwebby, Evanescent, Flaring, Large, None, Pendant, Sheathing, Zone}
(20) Spore-print-color	{Black, Brown, Buff, Chocolate, Green, Orange, Purple, White, Yellow}
(21) Population	{Abundant, Clustered, Numerous, Scattered, Several, Solitary}
(22) Habitat	{Grasses, Leaves, Meadows, Paths, Urban, Waste, Woods}
(23) Class	{Edible, Poison}

Table 4-10. All attributes in Mushroom dataset.

Of these attributes, there are 30% missing values in one of them. Categorical attributes are converted into binary attributes and expanded from 22 attributes to 121 attributes. The dataset used in our experiment is relatively large. The attributes also take on many possible values of which some are likely to be irrelevant. The result of the mushroom dataset is illustrated in Table 4-11 below.

	Accuracy	Rank
GACDD	89.8%	1
K-means	78.5%	4
K-modes	73.7%	5
COBWEB	87.3%	3
GA clustering	62.3%	6
Autoclass	89.0%	2

Table 4-11. Accuracy for the mushroom dataset.

In the experiment, our algorithm performed better than the others. Since each attribute in the data set can take on many different values, there are high probabilities that some of them are irrelevant. From the experiment, those algorithms that rely on the use of conditional probability measure performed relatively better as they are better able to handle noise. The exception to this is the GA-based clustering algorithm which is based on an impurity measure. When the attributes can take on many values and the number of clusters is comparatively small, each cluster may be regarded as impure by its own as each attribute in the cluster may be required to take several values. It is difficult to define a good cluster by such similarity measure. From such point of view, the similarity measure is important for the definition of a good cluster.

The results of *K*-means and *K*-modes are also rather low compared to the proposed algorithm. The limitations of these algorithms in finding global optimum are a great barrier to obtaining better results. COBWEB and Autoclass, also did not perform too well due perhaps to the distance measure not being able to handle categorical data containing too many possible values.

Except for the proposed algorithm, the other clustering algorithms do not perform too well with this set of data. This is likely due to relatively more irrelevant attributes in this set of data. It is also because some attribute-value pairs in some attributes always seem to contradict with most possible groupings of the records. The overall contribution of these attributes to the similarity measures may take some bad effect on the overall score. And this can result in a drop in overall accuracy. For the proposed algorithm, since it is better able to identify irrelevant attributes, it performed better. For example, in this mushroom dataset, we discovered that the stalk-shape and veil-type are irrelevant to the final grouping of the result since no interesting rules are discovered for these two attributes. As a result, the clusters discovered by the proposed algorithm are more meaningful.

4.2.7. All datasets comparisons

In the experiments, we compared our algorithm with various well-known algorithms. In order to determine the robustness of the algorithm, The data sets we used, as described above, has these characteristics: (i) they contain noisy and missing values; (ii) the clusters are of unequal sizes; (iii) the cluster numbers are relatively large; (iv) the data set are relatively large and (v) they are data sets with relatively many attribute value-pairs. The experimental results are summarized in Table 4-12.

	GACDD	*K*-means	*K*-modes	COBWEB	GA-based clustering	Autoclass
Synthetic	***100% (1st)***	66.0% (6^{th})	72.8% (5^{th})	***100% (1st)***	93.0% (4^{th})	***100% (1st)***
Synthetic 10% noise	***98.5% (1^{st})***	84.3% (5^{th})	74.5% (6^{th})	95.7% (2^{nd})	90.2% (4^{th})	94.2% (3^{rd})
Synthetic 20% noise	***88.6% (1^{st})***	76.2% (4^{th})	73.2% (5^{th})	80.5% (3^{rd})	62.1% (6^{th})	86.6% (2^{nd})
Synthetic 50% noise	80.6% ***(1^{st})***	31.2% (5^{th})	30.7% (6^{th})	70.3% (3^{rd})	51.8% (4^{th})	74.2% (2^{nd})
Soybean	***100% (1st)***	84.5% (6^{th})	88.9% (5^{th})	***100% (1st)***	94.5% (4^{th})	***100% (1st)***
Zoo	***92.1% (1st)***	74.7% (6^{th})	75.7% (5^{th})	89.6% (2^{nd})	78.3% (4^{th})	87.1% (3^{rd})
Vote	89.3% ***(1st)***	83.6% (6^{th})	86.4% (4^{th})	***88.0%*** (2^{nd})	86.4% (4^{th})	87.3% (2^{nd})
Heart	***80.1% (1^{st})***	72.6% (4^{th})	69.1% (5^{th})	76.0% (2^{nd})	65.7% (6^{th})	74.5% (3^{rd})
Mushroom	***89.8% (1^{st})***	78.5% (4^{th})	73.7% (5^{th})	87.3% (3^{rd})	62.3% (6^{th})	89.0% (2^{nd})

Average	***91.0%*** ***(1st)***	72.41% (6th)	71.66% (5th)	87.49% (3rd)	76.03% (4th)	88.1% (2nd)
S.D.	0.1073	0.1923	0.1977	0.1037	0.1074	0.1202

Table 4-12. Summary of the experimental results.

Table 4-12 contains a summary of the experimental results for all data sets. Among all these algorithms, the proposed clustering algorithm performed the best. The average correct classification rate of 91.0% is statistically significantly better than the average and than that of Autoclass and COBWEB that is the second and third bests.

From the above table, it may give one an impression that the top three algorithms are much better than the bottom three. While this is the case with the average classification accuracy measure, it shall be noted that, among the top three, COBWEB can fluctuate greatly in accuracy. Moreover, since only COBWEB cannot preset the desired number of clusters, there exists some trials without enough clusters found, and this induced a large drop in accuracy. The small standard derivation in our algorithm shows that it is a reliable clustering algorithm.

4.3. Discussions

The experimental results show that the proposed algorithm can be very effective. This can be due to the differences in characteristics in the proposed algorithm when compared to the others. The details are given in Table 4-13.

	GACDD	*K*-means	*K*-modes	COBWEB	GA-based Clustering	Autoclass
1) Similarity Measure	Model	Dist.	Dist.	Model	Model	Model
2) Method Type	Partitioning	Partitioning	Partitioning	Hierarchical	Partitioning	Partitioning
3) Missing Value	✔			✔	✔	✔
4) Irrelevant Features	✔				✔	✔
5) Result Interpretation	✔			Certain extent		Certain extent
6) Capability for Fuzzy clustering	✔					✔

Table 4-13. Comparison of the algorithms' characteristics

The results of the classical *K*-algorithms, such as *K*-means and *K*-modes in the experiments are not very well and they are often affected by the initial settings of the cluster centres. Since there is no mechanism for these methods to hop out of a local

optimum, the grouping of the data records is sensitive to initial centre selection. Also, if the clusters are not spherical and not of convex shape or the clusters size varies greatly, the performance of these algorithms are usually relatively poor.

Comparing with *K*-means, it should be noted that *K*-modes should perform better as it is supposed to better handle categorical data. On the contrary, *K*-means need to pre-process the data by making those categorical attributes binary. From the experiment, we discovered that the results of these two methods are more or less the same. However, the fluctuation in accuracy of *K*-means is much larger (when noise and missing values are at 20% and 50%). This may be due to the loss of information after transformation. The reason behind it is that *K*-means uses spatial distance to determine the differences between two records. When the initial centers are not selected well, the spatial distance similarity measure distorts the meaning of the cluster centers and assignments of the following records will be determined wrongly and the cluster centers are updated wrongly in a vicious circle. Because of the distance measure adopted and their lack of features to distinguish between relevant and irrelevant attributes, *K*-means and *K*-modes do not usually cater well for noisy and missing data. Basically, whenever there are missing values, they will be replaced either by the mean or mode value. For some data set, this can distort the original data set and can lead to poor performance [Matthews and Hearne 1991].

Compared to other clustering algorithms, COBWEB is based on a hierarchical approach. COBWEB performs quite well and achieves relatively high accuracy (ranked 3rd with 87.49% accuracy). The probability similarity measure makes COBWEB resistant to missing values. Compared to *K*-means and *K*-modes, COBWEB performs better as it adopts a probability measure as a similarity measure and this makes it more resistant to missing values and noise.

Since COBWEB generates clusters in a hierarchical manner, many possible groupings can be discovered for a given set of data. In hierarchical clustering, the number and grouping of clusters are determined by cutting a dendrogram at a particular height (sometimes termed the best-cut). For example, from Figure 4-3a, we

can see that two clusters are formed by cutting at (1) while four clusters are formed by cutting at (2). If we want to obtain three clusters, since only two and four clusters can be formed by cutting at (1) and (2) respectively, there is no choice to obtain the desired number of clusters, say three. In that case, we can only split the clusters of {A, B} or {C, D} by (3) and (4) respectively.

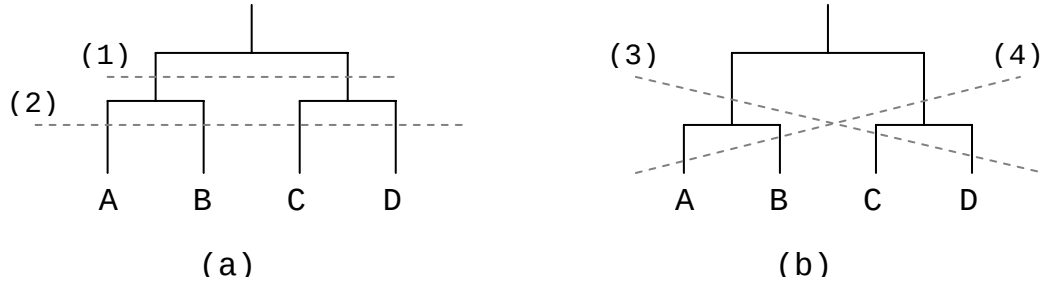


Figure 4-3. Partitioning in a hierarchical tree

For COBWEB, given a tree structure of clusters, we are facing two issues – how many clusters and which solution. In the experiment, we match the solutions against the clusters to obtain the greatest possible matches by merging and splitting different branches to get the desired number of clusters. However, it is very difficult to determine the splitting in real-life situation without knowing the class labels. Besides the above, COBWEB is sensitive to the input order of records. The tree structure may be totally different by different input order. In our experiment, we can easily handle it by matching the class label to obtain the greatest matches but it is not practical in real life.

Like GACDD, the GA-based clustering algorithm is also based on the use of GA. Unlike the *K*-algorithms, which search for the best cluster arrangement by hill-climbing, GA adopts a probabilistic search strategy which makes it better able to avoid hitting local optimum. The solutions that can be obtained can theoretically be globally optimal. In the GA-based clustering algorithm, it adopts a scaled entropy function as the similarity measure. The function calculates the similarity of an attribute-value partition relative to the class partitioning and vice versa. In fact, the GA-based clustering algorithm consists of two phases: the first phase extracts a small sample to calculate the importance of the attributes contributing to the final

clustering whereas the second phase use the relevant attributes for obtaining the result. However, in the first phase, it cannot determine if an attribute is relevant or irrelevant. It leaves the user to determine whether to include the attributes in the second phase for clustering. In the presence of noise or irrelevant attribute, the similarity function seems to be adversely affected and not be able to accurately reflect how good a clustering arrangement is.

Autoclass imposes a classical finite mixture distribution model on the data and uses a Bayesian method to derive the most probable class for the records giving prior information. The experimental results show that the accuracies in different datasets are also quite high. Many studies have reported that the performance of Bayesian inference is satisfactory. The experimental results show Autoclass handle data with missing value quite well. Autoclass can also identify the influence of the attributes to the final clustering. Despite its good performance, the computational complexity required by probability estimation is extremely high and it is attributable to the exhaustive search approach.

A disadvantage with Autoclass is that the time taken increases rapidly with the dataset size. In addition, Autoclass also suffers from the over-fitting problem associated with the maximum likelihood optimization methods for probabilistic model [Li and Biswas 2002]. That is, Autoclass can be heavily biased by the existence of small clusters and produce too many partitions. From Li's result and from our experiments, it still shows that Autoclass produces excessive clusters. Since we pre-defined the number of clusters in Autoclass, the shortcoming of producing excessive clusters does not reflect in the above experiment. Besides, the maximum likelihood approach in Autoclass can be sensitive to the choice of starting values.

Compared to the others, the proposed GA based clustering algorithm GACDD, performs better in many way than the compared algorithms in the experiment. Our algorithm used a proximity measure that is quite different. Though if-then interesting patterns for clustering have been studied in some papers, the application of the patterns suffers from practical difficulties. It required user to determine a threshold

such as minimum support and confidence to derive the interesting patterns [Han *et al.*1997]. However, it is a difficult task for normal user to define the threshold. If it is set too high, a user may miss some useful rules but if it is set too low, the algorithm may be overwhelmed by many irrelevant attributes. The interesting patterns generated are a statistically significant one in our algorithm. The rules will only be considered interesting if they are statistically significant. Hence, the interesting patterns generated are data-driven. Interesting patterns help a lot in cluster interpretation and irrelevant attributes identification.

All traditional distance based clustering algorithms cannot generate explanations for their clustering results. For data mining task to be effective, data interpretation is one of the vital processes in KDD [Fayyad *et al.* 1996a, 1996b, 1996c]. COBWEB gives explanation to a certain extent by the conceptual tree construction, while other compared algorithms only provide the representatives of the clusters for interpretation. On the contrary, our algorithm gives a good result interpretation. It not only provides the result with the interesting patterns classifying a record to the cluster, but also a measure of how important the patterns contribute to it. Since the patterns describe the relationship and importance of an attribute-value to a cluster label, it is very clear to the user to make further investigation especially in marketing analysis and customer relationship analysis.

Some features are irrelevant and they provide little correlation with other features. Clustering by the values of these few irrelevant attributes will ruin the grouping in many other attributes. As a result, it is better to ignore those “irrelevant” features, as they will produce noise in the clustering. Many studies show that relevant feature selection can improve accuracy [Devaney and Ram 1997; and Talavera 1999]. Our algorithm generates rules in a statistically significant manner; for those attributes-values that are not statistically significant, it will be regarded as not important, and hence irrelevant. Feature selection improves performance with respect of classification accuracy. Moreover, it can greatly reduce the data dimensionality, thus improving speed. Also, it helps user to identify these irrelevant attributes and assist in result interpretation.

Our algorithm maintains its accuracy in the presence of the noise and missing values. The experimental results indicate that it also performs well in large dataset, dataset with uneven cluster and dataset with many clusters.

GACDD employs the novel genetic operators and contains two phases' operations. In the first phase, it is the normal genetic operations of crossover and mutation; while in the second phase, it is a rule-based reclassification. The significance of reclassification is two-fold. The first function for reclassification is to rectify the bad gene produced in genetic operators and the deficiency of genetic algorithm to get the global optimum. During genetic operation, mutation will change some genes in order to introduce diversity in the population. However, it may change some good genes into bad genes. Though we can eliminate these by not allowing such mutation after calculating the fitness, normally we still allow these to exist to introduce enough diversity to avoid premature convergence. In another case, mutation change the good gene into bad gene, however, it simultaneously changes some bad gene into good one, so the good one compensates for the error introduced. If the total number of good and bad gene remains unchanged, it will not decrease in fitness and there is no mechanism to prevent such an error. Another thing is GA can easily reach its local optimum, but after that it may take considerable time to get its global maximum. The time taken may be much longer than that for local optimum. Consider a large dataset with one million records, after certain time, it gets all correctly classify into right cluster except one record. It is very difficult to find the error record and then can change correctly to the desire cluster. So, it is hardly surprising the result of genetic algorithm always contains some errors. The reclassification in such case shows its power. When GA reaches its local optimum, it always gets most records in correct cluster. By making use of the interesting rules hidden in these records, we can put back the error records into the correct cluster. The second function of reclassification is that correct clustering depends on the correct interesting patterns. It implies if we can get most interesting patterns, we can already get most records correct. In a genetic algorithm with one thousand iterations, the chromosome usually converges quickly and most interesting patterns will get correct in the early generations. By

using the patterns for reclassifying the records, we do not need to rely on genetic operations to change the gene correctly by chance to get better clustering. In short, we can greatly shorten the clustering time and increase efficiency.

In conclusion, the performance of the clustering methods is limited by two factors: the suitability of the objective function and the effectiveness of the search method. For the algorithms examined, it is interesting that the results can be classified to two groups in terms of accuracy. Our algorithm, COBWEB and Autoclass is in the first group while *K*-means, *K*-modes and GA based clustering in the second group. The searching methods in *K*-algorithms limit them from obtaining the global solution. Though they can get satisfactory performance in accuracy and speed, the capability to obtain the best solution is unpleasant. GA based clustering solves the problem by genetic algorithm but the general entropy similarity measure makes it difficult to differentiate records and put them into different clusters. Our method also employ genetic algorithm. However, our algorithm that uses the interesting patterns make the data records easily be classified back into the correct cluster. Also, GACDD employs a totally different similarity function that groups the records based on the characteristics they process. The category utility and Bayesian inference of COBWEB and Autoclass also make it more effective in clustering.

Chapter 5

Clustering When K is Unknown

Traditionally, determining the optimal number of clusters when clustering a dataset is a difficult problem. Many clustering techniques, such as those based on optimization methods, like the *K*-means, require users to input the number of desired clusters. For hierarchical techniques, data records are merged together to form sub-clusters and sub-clusters are merged in a hierarchical manner according to some optimization criteria. In the end, it forms a single cluster containing all records. A tree called dendrogram is often used to describe which sub-clusters are merged together and the distance between them. In a dendrogram, we can form the desired number of clusters by cutting the tree at some level. The hierarchical techniques add certain flexibility to clustering, it is still very hard to determine the suitable cutting level however. While the estimate of a suitable number of clusters is relatively easy, for hierarchical methods that depend on distance measure such as conceptual clustering, it can be very hard to determine the cutting as no distance measure can be used as indicator. In practice, therefore, users are required to run experiments for different *k* and then decide for themselves how the results should be interpreted given different cluster numbers. Alternatively, cluster validation techniques can be used to help determine the optimal cluster numbers. These techniques are used to determine intra-cluster and inter-cluster similarity. The best number of clusters can be found by minimizing the intra-cluster distances while maximizing inter-cluster distances. There have been some works on determining the right number of clusters [Cristofor and Simovici 2002; Everitt *et al.* 2001; Fraley and Raftery 1998; and Tibshirani *et al.*

2000]. They are, however, not developed for discrete-valued data. For such data, we need to have a good technique to allow a suitable number of clusters to be determined.

5.1 Extended GACDD

To enhance the applicability to real world problem, GACDD is extended to find the optimal number of clusters for discrete data. In the extension, GA is modified in the following ways: (1) Chromosomes in the population can contain different cluster number in a range defined by minimum and maximum cluster number, (2) Chromosomes is initialised randomly to process different cluster numbers in a range specified, (3) Two-point crossover randomly selects two cutting points containing one or more clusters, (4) Empty groups will be deleted, (5) Two new mutation operators are induced – *merge* mutation and *split* mutation operators and (6) Fitness function is enhanced for new criteria. The encoding of the chromosome will be remained as using cluster as the gene. In the following sections, we explain in detail the extended GACDD for unknown *k*.

5.1.1 Minimum and Maximum Cluster Number

In GACDD, we need to specify *k*, the desired number of clusters. In the extended GACDD, the range, more exactly, the minimum and maximum cluster numbers are defined. For each GA operation, the cluster number contained in chromosome will be checked against the range. The main purpose is to prevent the GA operations from generating too few and too much clusters. Actually, we can arbitrarily define the lower and upper limit of cluster number in the initialisation of chromosome. Then, leaving GA operators to increase or decrease the number of clusters during GA operations. GA will not indefinitely increase the cluster number because cluster number exceeding the natural cluster will cause the chromosome less fit than other as it processes records with dissimilar characteristics or attribute-value pair in the cluster.

5.1.2 Initialization

For initialisation, we define the minimum and maximum desired cluster numbers. [Cole 1998] Actually, we can just define the maximum bound because when assign records randomly to each cluster, there may happen that some clusters will not be assigned any records, which result in different cluster numbers in the chromosome pool. However, since each cluster has equal chance being assigned, when the data set has many records, the chance that results in empty clusters will be rare. To guarantee the chromosome pool to have adequate chromosomes of different cluster numbers, we need to make sure similar amount of chromosomes of different cluster numbers inside. As a result, for each chromosome, first selected randomly the desired cluster number from the range defined in the upper and lower limit of clusters, say k_1 and k_2. If the lower and upper limits are 3 and 5 respectively, we can generate chromosome with 3 or 4 or 5 clusters. Each chromosome is randomly assigned the number of clusters from k_1 and k_2. After assignment, we can check whether the chromosome contains enough clusters and invalid chromosome will be deleted. The process repeated until all chromosomes contain their desired number of clusters.

```
1   proc initialize(↔Pt, ←S)      /* Pt is population and S is record set */
2       for i = 1 to |Pt| do      /* For each chromosome in the population */
3           /* Initialize the number of cluster randomly between minCluster and
                maxCluster */
4           K = rand(minCluster … maxCluster);
5           Pt [i] = set_list (K)    /* Initilize the list of list to contain K list */
6           while checkOK  do
7           /* Assign each record to the lists */
8               for q = 1 to |S| do k = rand(1 …K); add(S[q], Pt [i], k) od
9                   checkOK = valid_check(Pt [i]);
10          od
11      od
12
13  proc add(←s, ↔list,←k)
14      add_record(list[k], s)      /* Add record S[q] to list[k] */
```

Figure 5-1. Pseudo code for Initialization

5.1.3 Crossover

The crossover operation is modified to adopt a two-point crossover. Since the chromosomes employ a list-of-list representation, two-point representing the starting list and ending list is selected. The parent chromosomes undergo the two-point selection and exchange the selected groups. Figure 5-2 depicts the crossover process. This results in an opportunity of increasing and decreasing in the number of clusters presented in the chromosome. The number of clusters is then checked to examine whether it is still within a valid range. If not, the invalid child chromosomes will be thrown away, and the crossover will be repeated till it is accepted. There may exist duplicate genes in the crossover chromosome, the duplicate genes are deleted and the remaining one will be redistributed in the same manner as that in Chapter 3. In case that the cluster is empty after deletion, the cluster will be deleted from the chromosome.

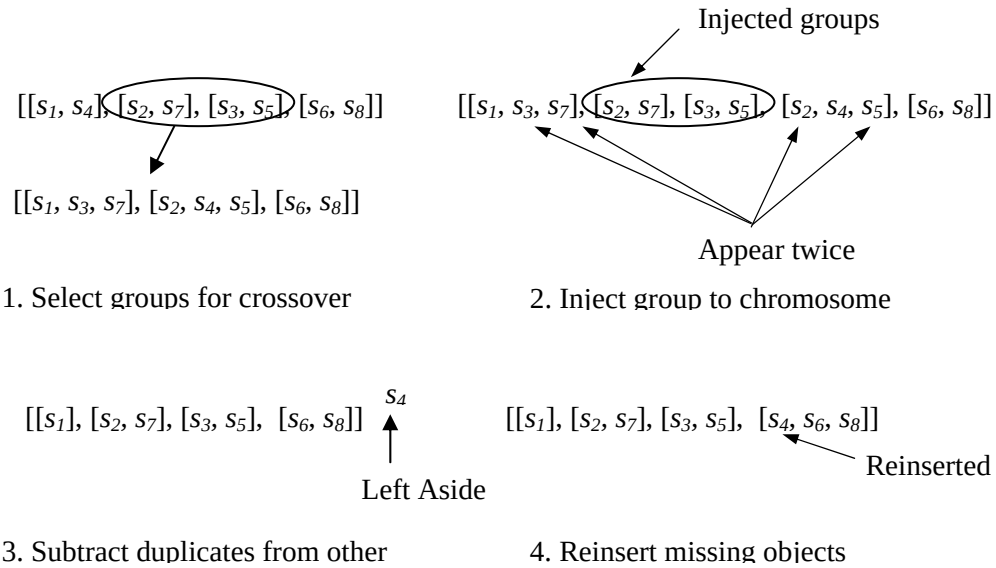


Figure 5-2. Two points crossover.

```
1  func crossover(←Pt[i], ←Pt[j], →list3, →list4)
2      index1 = rand(1…| Pt[i]|); index2 = rand(1…| Pt[j]|);
3      index3 = rand(1…| Pt[i]|); index4 = rand(1…| Pt[j]|);
4      /* Swap the lists between the two chromosomes */
5      list3 = merge(Pt[i], Pt[j], index1, index3, index2, index4);
6      list4 = merge(Pt[j], Pt[i], index2, index4, index1, index3);
7      return list1, list2;
8
9  func merge(←list2, ←list1, ←index1, ←index2, ←index3, ←index4)
```

```
10      for i =index1 to index2 do
11          add_list(list1[i], list2) od       /* Add the list to another list-of-list */
12      /* For each record in the list1 and list2 */
13      for m = index1 to index2 do
14          for n = index3 to index4 do
15              for i =1 to |list2[m]|
16               /* For each record in the m^th list1 */
17                  for j =1 to | list1[n]| if (list2[m][i]== list1[n][j]) do
    delete(list2[m][i]); od fi
18      for m = index1 to index2 do
19          for i =1 to |list2[m]|
20              roulettewheelreplace(list2, list2[m]); return chrom;
21
22  proc roulettewheelreplace(↔list2, ←list2[index1])
23      for i = 1 to |list2|
24          prob[i] =1/|list2 [i]|
25      prob[i] = prob[i]/sum(prob[])
26      for (i = 1 to |prob[]|) prob_total= prob_total + prob[i];  prob[i] =
    prob_total;
27      for j = |list2[index1]| do
28          k = rand(0…1);
29          for i = 1 to |prob[]| if k <  prob[i] do add(list2[i], record) od fi
30      od
```

Figure 5-3. Pseudo code for Crossover

5.1.4 Mutation

In addition to the swap mutation described in Chapter 3, two new mutation operations – split mutation and merge mutation are added. It then results in three types of mutation. The swap mutation switches the genes between two groups while keeping the total number of groups equal. It is the process that randomly picks up two groups in the chromosomes, then randomly picks the gene from each group and swaps the genes between groups. The split and merge mutation increase or decrease the number of clusters in the chromosome. The split mutation operates by randomly taking out a cluster. Larger cluster has a greater chance to be chosen. Then, the cluster is split into two. Records in the cluster are randomly assigned to either one of two split clusters with equal chance. The split will result in two clusters with similar size. The merge mutation runs by randomly choosing two clusters in the chromosome, then merge the records together as a single cluster. The splitting and merging of the chromosome is determined by two parameters – split and merge mutation rates which are defined by the user. Since frequent split and merge

mutation will spoil the swap mutation process, they are expected to occur occasionally.

```
1   proc mutate(↔list)
2       swap_mutate(list);
3       merge_mutate(list);
4       split_mutate(list);
5
6   proc swap_mutate(↔list)
7       index1 = rand(1… |list|); index2 = rand(1…|list|);
8       mutate_index = rand(1…|list[index1]|);
9       add_record(list[index2], list[index1][mutate_index]);
10      delete_record(list[index1][mutate_index]);
11
12  proc merge_mutate(↔list)
13      merge_index1 = rand(1…|list|);
14      merge_index2 = rand(1…|list|);
15      for i = 1 to |list[merge_index1]| do
16          add_record(list[new], list [merge_index1] [i])
17      od
18      for i = 1 to |list[merge_index2]| do
19          add_record(list[new], list [merge_index2] [i])
20      od
21      delete(list, list[merge_index1]);
22      delete(list, list[merge_index2]);
23      add(list, list[new]);
24
25  proc split_mutate(↔list)
26      mutate_index = rand(1…|list|);
27      for i = 1 to |list[mutate_index]| do
28          if rand(0…1) < 0.5
29              add_record(list[new1], list [mutate_index] [i])
30          else
31              add_record(list[new2], list [mutate_index] [i])
32      od
33      delete(list, list[mutate_index]);
34      add(list, list[new1]);
35      add(list, list[new2]);
```

Figure 5-4. Pseudo code for Mutation

5.1.5 Fitness Function

Since the chromosomes contain different cluster numbers, the fitness function is slightly modified to have a fair evaluation. In a chromosome, more clusters tend to generate more rules and hence, the total fitness for the chromosome with more

clusters must be greater. To give a fair evaluation, the total fitness should be divided by the number of clusters presented in that solution. Since the original fitness is measured in an entropy approach, the cluster number will also take logarithm as the entropy one. As a result, the fitness function is changed to the following:

$$fitness = \sum_{p=1}^{P} \frac{\sum_{i=1}^{N} W_{t_i c_p}}{no.\ of\ records\ in\ C_p} / \log_2(no.\ of\ clusters)$$

5.2 Experimental Results

Experiments are set up to examine the effectiveness of extended GACDD to find the natural number of clusters. The natural number of clusters is the number in the class labels in the data sets. Several synthetic and real-life datasets will be used for experiments. The parameters of GACDD are set as follow. The population is 100. The maximum iteration is 10000. The crossover rate is 0.06. The swap, split and merge mutation rate will be 0.02, 0.02 and 0.01 respectively. The minimum and maximum numbers of clusters are set to be 2 and 10 accordingly.

We examine the GACDD with 1) Synthetic dataset; 2) Soybean dataset; 3) Vote dataset and 4) Zoo dataset. Detail description for the dataset can be found in Chapter 4. We run 1000 trails for each datasets in the experiment. The experimental result is illustrated in Table 5-1.

Dataset	Cluster number in dataset	GACDD	
		Cluster number	Accuracy
Synthetic	4	4	100%
Soybean	4	4	100%
Vote	2	2	87.13%
Zoo	7	4	83.16%

Table 5-1. Comparison of accuracy and number of clusters by our algorithm

In the synthetic and soybean dataset tests, extended GACDD can discover correctly the number of clusters. Moreover, they also get 100% accuracy in clustering the records. For extended GACDD, although GA sometimes finds more or less clusters than the desired number, the reclassification shows its capability to correct it to the right number of clusters. In fact, during the genetic operations, the split and merge

mutation may introduce more or less clusters. However, the reclassification will put back those wrongly classified records to the correct cluster according to the interesting patterns obtained. This result in some clusters empty of records. The empty cluster will be deleted. So the correct number of cluster can be found.

For the vote dataset, extended GACDD discovers correctly the natural number of clusters. Moreover, we can get 87.13% accuracy in clustering the records correctly. Table 5-2 shows the result of extended GACDD.

Cluster	**1**	**2**
Democrat	220	47
Republican	9	159

Table 5-2. Clustering of vote datasets by our algorithm

For the zoo dataset, extended GACDD only finds 4 clusters. The most significant error happens in the Amphibians, Insects and Invertebrates. These 3 categories are merged into one big cluster. Actually, reptiles show evidence that it shares many common features with those of other clusters.

Cluster	**1**	**2**	**3**	**4**
Mammals	41	0	0	0
Birds	0	0	0	20
Reptiles	2	3	0	0
Fishes	0	13	0	0
Amphibians	1	0	3	0
Insects	0	0	8	0
Invertebrates	0	0	10	0

Table 5-3. Clustering of zoo datasets by our algorithm

From the experimental results, extended GACDD demonstrates capability on finding the number of clusters. Most of the time, it can find the cluster number correctly with high classification accuracy. Extended GACDD controls the generation of excessive clusters by two mechanisms. We can increase the merge mutation rate to allow greater possibilities for the cluster to merge together. Also, the reclassification can group back the records according to the interesting patterns obtained. As a result, we can reduce the number of clusters formed.

Chapter 6

Conclusion

In the thesis, we introduce a novel clustering method for categorical data. Basically, the effectiveness of a clustering method is critically determined by two factors, the searching method and the proximity criteria. The proposed algorithm employs a genetic algorithm for clustering that is shown in the experiments to be an effective clustering method for categorical data. The proximity criteria adopt a rule-based information theoretical measure called weight of evidence. It finds the interesting patterns and measures the weight of these patterns that supporting the presence of an attribute-value pair to be relevant to a cluster label. By summing up the total weight that the records acquire in the patterns due to presence of both the attribute-value and the corresponding cluster label, we can measure the fitness in the chromosome and hence how best the records are clustered together.

Traditional genetic algorithm has difficulties in solving clustering problem. The difficulties are in both chromosomes encoding and genetic operations. The proposed algorithm is modified to handle such problem. The proposed algorithm involves a two-phase process. In the first phase, it is a genetic-based searching for the meaningful grouping of the data. In the second phase, it applies those interesting patterns to reclassify the records in the dataset. It is changed from records to cluster as the building block for encoding, crossover and mutation.

Empirical results show that the proposed algorithm can successfully find the clusters in the datasets. The results illustrate promising performance and it is the most accurate one among the compared algorithms. In addition to the encouraging performance, the proposed clustering algorithm is also able to discover a set of if-then rules describing the pattern underlying all discovered clusters. We apply simulated data as control experiment to test the capability to find out those rules. We defined a set of rules embedded in the simulated data. The result indicates that the proposed algorithm can successfully discover the rules hidden in the data. From user point of view, the knowledge can be unveiled and interpreted for further analysis.

The proposed algorithm brings a number of advantages. It can handle data with noisy and missing values. It can be easily modified to handle huge amount of data by using data sampling. It can identify the irrelevant attributes and ignore those attributes in the clustering to reduce dimensionality. It can uncover the underlying knowledge in terms of interesting patterns with weight assigned. Most importantly, it can give an easy interpretation.

For years, finding the natural number of clusters in a dataset has been a difficult issue in many clustering algorithms. It is not easy for user to determine the desired number of clusters. Even a hierarchical tree of clusters is given; it is still hard to determine the cut points. The proposed algorithm is modified in Chapter 5 to find the optimal number of clusters. It is modified in several ways. It allows a variable number of clusters in the chromosomes. It introduces split and merge mutation and new crossover operations. In addition, the fitness function is enhanced to give a fair evaluation. The experimental results illustrate the proposed algorithm can successfully find the correct number of clusters in many datasets which is also not an easy task for many clustering methods. Though the proposed algorithm may produces excessive clusters but it has ways to rectify it. It is by controlling the merge mutation rate and effect of reclassification. As a result, the number of cluster can be carefully controlled.

6.1 Future Work

Empirical results show our algorithm can handle categorical data and is comparable to many common clustering algorithms. There are still areas for further enhancement. Future work can be done on the following areas:

– Controlling of GA parameters and operations

– Defining the interesting rules

– High dimensional data and huge data size

6.1.1. Controlling of GA parameters and operations

Empirical and theoretical results show that the selection of parameters used in GA can have a large impact on the performance of the GA. In [Grefenstette 1986], Grefenstette proposed a set of control parameters that is largely adopted by domain users. Empirically, if the mutation rate is too small, it may take a long time to find an optimal solution and it is set too high, the convergence rate can be slow. Population size and crossover rate can affect the diversity. If the crossover rate is too small, it will lead to small diversity and early convergence of the evolutionary process. If it is set too large, it can eliminate easily the best chromosomes. Moreover, the different methods in genetic operations such as selection, mutation and crossover can also have some influence on the results obtained at the end of an evolutionary process. As the interaction of the control parameters and the performance of a GA are known to be complex, many methods have been proposed to solve the problems [Back 1992; Srinivas and Patnaik 1994; and Lee and Tagaki 1993]. In [Back 1992; and Srinivas and Patnaik 1994], it employs a self-adaptive system to control the parameters. That is, the probabilities of crossover and mutation are varied depending on the fitness values of the solutions. Fuzzy approach is also applied to solve the problem. In [Lee and Tagaki 1993], a fuzzy knowledge-based system is used to dynamically control GA parameters. The results demonstrate that the performance in accuracy and speed is much improved.

6.1.2. Defining interesting rules

Many GAs generate initial populations randomly. For GACDD, we also assign records to the cluster randomly initially. The initialization can affect the genetic search in our algorithm in the following way.

GACDD detects for interesting attribute values and uses weight of evidence measure as a fitness measure. In the initialization, there may not be enough statistically significant interesting patterns in the chromosomes. This, as a result, leads to the little variances in the fitness of the chromosomes and hence small diversity in the population and require more iterations to get the solution.

For some datasets, the separation among clusters may not be very big and there may be severely overlapping clusters. Normally, we can increase the number of relevant attribute values discovered by reducing the confidence level when detecting for significant statistics. We can lower it, say from 99% to 95%. The increase in the detected values will increase the diversity in the clusters but will lower the robustness when testing for the cluster re-classification accuracy.

The initialization process, it should be pointed out, does not have to be completely random. In fact, if we use some fast algorithms, such as the K-modes, to initialize the clusters for the initial generation of chromosomes, the discovery of relevant attributes can be much faster and this can result in the evolutionary process being terminated earlier.

6.1.3. High dimensional data and large datasets

The kind of problems that GA solves is typically optimization problems and the search space increases exponentially as a function of the problem size. For this reason, the number of generations it takes for a globally optimum solution to be found increases rapidly with the problem complexity. A number of methods have been described in the literature [Cantú-Paz 1998; Ratha et al. 1995; Zaki et al. 1997; and Toivonen 1996] to address this problem and among them, the use of parallel computation approaches in GA and the use of data sampling to improve convergence seem to be more popular.

Parallel GA has been investigated for a number of years [Cantú-Paz 1998; and Ratha *et al.* 1995]. The basic idea behind many parallel GA is to divide a task into chunks

and solve the chunks simultaneously using multiple processors. This divide-and-conquer approach can be used in different ways and this leads to different methods to perform parallel GAs. Some methods change the behaviour of the algorithm while others do not. There are typically three ways to parallelize GAs: global parallelization, coarse-grained parallel GAs or distributed GAs and fine-grained parallel GAs. Early studies on parallel GAs was made by Grefenstette [Grefenstette 1981]. The prototype is a global parallel GA with a “master” processor that does selection and applies crossover and mutation. The individuals are sent to “slave” processors to be evaluated and returned to the master at the end of every generation. The investigation shows that the speed-up in time is nearly linear-scaled with the number of processor [Talbi and Bessière 1991].

Data sampling is a technique largely applied to the field of statistics to model the characteristics of the population. For large databases, data sampling can be used in such cases to sample some of the records for mining. The effectiveness of the result can be largely influenced by the size of the sample and the sampling techniques but a major benefit of sampling is the speed and efficiency of working with a smaller data size that still contains the essence of the entire database. Sampling enables analysts to spend relatively more time fitting models and thereby less waiting time for modelling results. It also enables easier visualization of the data. For this reason, the use of sampling in data mining as statistically valid practice for processing large databases can be considered for future work [Zaki *et al.* 1997; and Toivonen 1996].

In data mining, the sampled data is used to construct predictive models, which in turn, are used to make prediction for the entire database. Therefore, the validity of a sample, that is, whether the sample is representative of the entire database, is critically important. Whether a sample is representative of the entire database is determined by two characteristics –the quality of the sample and the size of the sample. In order to find a sample that is good quality, a sample should be unbiased enough to be at least typical of the database, that is, each record has the same chance of being selected. A simple random sampling is the most typical sampling technique which a certain number of records is taken out from the whole population entirely by

random. Theoretically, if the sample size is large enough, the information in the population can also be maintained in the sample as each record is picked up by random. As a result, sample size is a major factor that will affect the outcome of the mining result.

Data sampling largely speeds up the mining process without compromising the result accuracy. However, for data sampling to be applied in clustering, the algorithm must be able to generate result with some cluster representatives or some rules in order to deduce the remaining data points in the population. Many clustering algorithms like hierarchical approach, concept learning approach, density-based approach and neural-net approach often hardly applied the data sampling. They required further processing, such as decision tree building from the sampling result, to classify the population. However, decision tree prediction based on sampling data may distort the original grouping of the dataset. Among the selected algorithms for comparison, that is *K*-means, *K*-modes, COBWEB, GA based clustering, Autoclass and GACDD, in the chapter 4, COBWEB and GA based clustering cannot naturally apply the data sampling without relying on another algorithm for predicting the population. In order to predict the records, GACDD is slightly modified to handle the sample data. GACDD will initially sample a small subset of records for deriving interesting patterns. GACDD then operates on the sampled data to find out the patterns by performing the GA operations and fitness functions. Followed by exploring the interesting patterns found on the sample data, GACDD then classify the entire data set to the corresponding cluster by the interesting patterns obtained from the sampled data.

Referencess

Agrawal, R., Gehrke, J., Gunopulos, D. and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. *Proc. of 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 98)*, Seatle, WA, June, pp. 94-105.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. of 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 93)*, Washington D.C, pp. 207-216.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., and Verkamo, A. (1996). Fast discovery of association rules. U.Fayyad, G. Piatetsky-Shapiro, P. Symth, and R. Uthurusamy (1996), editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, pp. 307-328.

Agrawal, R. and Srikant, R. (1994). Fast algorithm for mining association rules. *Proc. of the 20th VLDB Conf.*, pp. 487-499.

Anderberg, M.R. (1973). *Cluster analysis for applications.* Academic Press, New York.

Anderson, J. and Matessa, M. (1991). An incremental Bayesian algorithm for categorization, *Concept Formation: Knowledge and Experience in Unsupervised Learning*, edited by Fisher, D., Pazzani, M. and Langley, P.. Morgan Kaufmann Publisher, Inc., San Mateo, California, pp. 45-70.

Ankerst, M., Breunig, M., Kriegel, H. and Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. *Proc. of 1999 ACM-SIGMOD Int. Conf. Management Data (SIGMOD' 99)*, Philadelphia, PA, June, pp. 49-60.

Back, T. (1992). Self-adaptation in genetic algorithms. In F. Varela and P. Bourgine, editors, Towards a Practice on Autonomous Systems: *Proc. of the 1st European Conf. on Articial Life*. MIT Press, pp. 45-70.

Ball, G. H. and Hall, D. J. (1967). A clustering technique for summarizing multivariate data. *Behavioral Science*, vol.12, pp. 153-155.

Berkhin, P. (2002). *Survey of Clustering Data Mining Techniques*. Accrue Software, Inc.

Bezdek, J., Boggavarapu, S., Hall, L. and Bensaid, A. (1994). Genetic algorithm guided clustering. *IEEE World Congress on Computational Intelligence in Proc. of the first IEEE Conf. on Evolutionary Computation*, vol. 1, pp. 34-39.

Bhandari, I. *et al*. (1997). Advanced scout: data mining and knowledge discovery in NBA data. Summary note. *Data mining and knowledge discovery*. 1 (1).

Biswas, G., Weinberg, J. and Fisher, D. (1998). ITERATE: A Conceptual Clustering Algorithm for Data Mining. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 28, no. 2, May.

Blake, C. and Merz, C. (1998). *UCI Repository of machine learning databases*. http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science.

Bobrowski, L. and Bezdek, J. C. (1991). C-Means clustering with the l_1 and l_∞ norms. *IEEE Transactions on Systems, Man and Cybernetics*, vol. 21 (3), pp. 545-554.

Brachman, R., Khabaa, T., Kloesgen, W., Piatesky-Shapiro, G. and Simoudis, E. (1996). Industrial applications of data mining and knowledge discovery. *Communication of the ACM*, 39(11), November.

Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10 (2), pp. 141-171.

Chan, K.C.C. and Au, W.H. (2001). Mining fuzzy association rules in a database containing relational and transactional data. *Data Mining and Computational Intelligence*, NY: Physica-Verlag, New York, pp. 95-114.

Chan, K.C.C. and Chung, L.L.H. (1999). Discovering clusters in databases containing mixed continuous and discrete-valued attributes. *Proc. of SPIE AeroSense'99 Data Mining and Knowledge Discovery: Theory, Tools, and Technology*, 5-9 April, pp. 22-31.

Chan, K.C.C., Wong, A.K.C. and Chiu, D.K.Y. (1988). OBSERVER: a probabilistic learning system for ordered events. *Pattern Recognition*, edited by J.Kittler, pp.507-517, Springer-Verlag, London, United Kingdom.

Chan, K.C.C. and Wong, A.K.C. (1990). APACS: a system for automated pattern analysis and classification. *Computational Intelligence*, vol. 6, pp. 119-131.

Cheeseman, P. and Stutz. J. (1995). Bayesian classification (Autoclass): theory and results. *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. PiatetskyShapiro, P. Smyth, R. Uthurusamy (eds.), Cambridge, MA: AAAI/MIT Press, pp. 153--180, 1996.

Cheeseman, P., Kelly, J., Self, M., Stutz. J., Taylor, W. and Freeman, D. (1988). A bayesian classification system. *Proc. of the Fifth International Conference on Machine Learning*, Ann Arbor; MI: Morgan Kaufmann, pp. 54-64.

Chen, S. (2002). *Genetic algorithms and genetic programming in computational finance*, Boston, MA: Kluwer.

Chung, L.L.H. (1999). *Discovering clusters in databases using an evolutionary approach*. Mphil. Thesis, Department of Computing, The Hong Kong Polytechnic University.

Cole, R.M. (1998). *Clustering with genetic algorithms*. Master Thesis, University of Western Australia.

Cover, T and Thomas, J. (1990). *Elements of Information Theory*. John Wiley & Sons, New York, NY.

Cristofor, D. and Simovici, D. (2002). An information-theoretical approach to clustering categorical databases using genetic algorithms. *SIAM ICDM*, Workshop on clustering high dimensional data.

Davis, L. (1991), *Handbook of Genetic Algorithms*. Van Nstrand Reinhold, New York.

DeJong, K. (1988). Learning with genetic algorithm: an overview. *Machine Learning*, vol. 3, pp. 121-138.

DeJong, K. and Spears, W. (1990). An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms. *Proc. of First Workshop Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, pp. 38-47.

Dempster, A. P., Laird, N.M. and Rubin, D.B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society*, vol. 39, pp. 1-38, 1977.

Devaney, M. and Ram, A. (1997). Efficient Feature Selection in Conceptual Clustering. Machine Learning. *Proc. of 14th International Conference on Machine Learning*, Nashville, TN, Morgan Kaufmann, pp. 92-97.

Dunham, M. (2003). *Data mining introductory and advanced topics*. Prentice Hall/ Pearson Education, New Jersey.

Ester, M., Kriegel, H., Sander, J. and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases. *Proc. of 1996 Int. Conf. Knowledge Discovery and Data Mining (KDD' 96)*, Portland, OR, Aug. , pp. 226-231.

Everitt, B., Landau, S. and Leese, M. (2001). *Cluster analysis.* 4th edition, Edward Arnold, New York.

Falkenauer, E. (1994). A new representation and operators for genetic algorithms applied to grouping problem. *Evolutionary Computation*, vol. 2(2), pp. 123-144.

Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley & Son, New York.

Fayyad, U. (1997). Data mining and knowledge discovery in databases: implications for scientific databases, *Proc. of 9th International Conference on Scientific and Statistical Database Management*, pp. 2-11.

Fayyad, U. (1998a). Taming the giants and the monsters: mining large databases for nuggets of knowledge. *Database Programming and Design*, vol. 11, no. 3, March.

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. MIT Press.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996a). Knowledge discovery and data mining: towards a unifying framework. *Proc. of Second Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996b). From data mining to knowledge discovery in databases. *AI MAGAZINE*, Fall, pp. 37-54.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996c). KDD for science data analysis: issues and examples. *Proc. of Second Int. Conf. on Knowledge Discovery and Data Mining (KDD-96)*, AAAI Press.

Fisher, D. (1987). Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, vol. 2 (2), pp. 139-172.

Fisher, D., Pazzani, M. and Langley, P. (1991). *Concept formation: knowledge and experience in unsupervised learning*, San Mateo, California, Morgan Kaufmann Publishers.

Fisher, D., Xu, L., Carnes, J., Reich, Y., S. Fenves, S., Chen, J., Shiavi, R., Biswas, G. and Weinberg, J. (1993). Applying AI Clustering to Engineering Tasks. *IEEE Expert*, December, pp. 51-60.

Fraley, C. and Raftery, A. (1998). How many clusters? Which clustering method? Answers via model-based cluster analysis. *Computer Journal*, 41, pp. 578--588.

Ganti, V., Gehrke, J. and Ramakrishnan, R. (1999). CACTUS – clustering categorical data using summaries. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA USA, Aug 15-18, pp. 73-83.

Gennari, J., Langley, P. and Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, vol. 40, pp. 11-61.

Gibson, D., Kleinberg, J. and Raghavan, P. (2000). Clustering categorical data: an approach based on dynamical systems. *Proc. of the 24th Annual International Conference on Very Large Databases (VLDB'98)*, VLDB Journal vol. 8, no.3-4, pp. 222-236.

Goldberg, D.E. (1989). Genetic algorithms in search, optimization and machine learning. Addison-Wesley.

Goldberg, D. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. G. Rawlins, editor, *Foundations of Genetic Algorithms*, Morgan Kaufmann, Berlin, pp. 69-93.

Goodall, D. (1966). A new similarity index based on probability. *Biometrics*, vol. 22, pp. 882-907.

Grefenstette, J. (1981). *Parallel adaptive algorithms for function optimization.* Technical Report. Nashville, TN: Vanderbilt University, Computer Science Department.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on System, Man, and Cybernetics*, SMC vol. 16(1), pp. 122-128.

Guha, S., Rastogi, R. and Shim, K. (1998). Cure: An efficient clustering algorithm for large databases. *Proc. of 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 98)*, Seattle WA, June, pp. 73-83.

Guha, S., Rastogi, R. and Shim, K. (1999). ROCK: A robust clustering algorithm for categorical attributes. *Proc. of the 15th ICDE*, 512-521, Sydney, Australia.

Han, E., Karypis, G., Kumar, V., and Mobasher, B. (1997). Clustering based on association rule hypergraphs. *SIGMOD'97 Workshop on Research Issues on Data Mining and Knowledge.*

Han, E., Karypis, G., Kumar, V., and Mobasher, B. (1998). Hypergraph based clustering in high-dimensional data sets: a summary of results, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 21, no. 1, IEEE Computer Society, March, pp. 15-22.

Han, J. and Kamber, M. (2001). *Data mining: concepts and techniques.* Morgan Kaufmann Publishers.

Huang, Z. (1997a). Clustering large data sets with mixed numeric and categorical values. *Proc. of First Pacific-Asia Conf. on Knowledge Discovery & Data Mining*, pp. 21-34.

Huang, Z. (1997b). A fast clustering algorithm to cluster very large categorical data sets in data mining. *DMKD.*

Huang, Z. (1998). Extension to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery 2(3)*, pp. 283-304.

Jain, A.K. and Dubes, R.C. (1988). *Algorithms for clustering data*. Prentice Hall.

Jain, A.K., Murty, M.N. and Flymn, P.J. (1999). Data clustering: a review. *ACM Computing Surveys*, vol. 31 (3), pp. 264-323.

Jiang, T. and De Ma, S. (1996). Cluster analysis using genetic algorithms. *Proc. of Int. Conf. on Signal Processing ICSP'96*, vol.2, pp. 1277-1279.

Jones, D.R. and Beltramo, M.A. (1991). Solving partitioning problems with genetic algorithms. *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 442-449.

Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1997). Multilevel Hypergraph Partitioning: Applications in VLSI Domain, *IEEE Trans. on Very Large Scale Integration Systems*, vol. 7, Mar., pp. 69-79.

Karypis, G., Han, E.H., and Kumar, V. (1999). CHAMELEON: A hierarchical clustering algorithm using dynamic modelling. *IEEE Computer: Special Issue on Data Analysis and Mining*, vol. 32, no. 8, pp.68-75.

Kaufman, L. and Rousseeuw, P. J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley & Sons.

Kemenade, C. (1996). Cluster Evolution Strategies. *Proc. of IEEE Int. Conf. on Evolutionary Computation 1996*, pp. 637-642.

Kennedy, R., Lee, Y., Roy, B., Reed, C. and Lippman R. (1997). *Solving data mining problems through pattern recognition*. The Data Warehousing Institute Series, Prentice Hall PTR, Upper Saddle River, New Jersey.

Krishna, K. and Murty, M. (1999). Genetic *k*-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, 29(3), Jun 1999, pp. 433-439.

Krovi, R. (1992). Genetic algorithms for clustering: a preliminary investigation, *Proc. of the Twenty-Fifth Hawaii Int. Conf. on System Sciences*, vol. 4, pp. 540-544.

Kuncheva, L. and Bezdek, J. (1998). Nearest prototype classification: clustering, genetic algorithm, or random search? *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, vol. 28, no. 1, February.

Lee, M. A. and Tagaki, H. (1993). Dynamic control of genetic algorithm using fuzzy logic techniques. *Proc. of the 5th Int. Conf. on Genetic Algorithms*, ICGA'93, pp. 76-83.

Leon, E., Nasraoui, O. and Gomez, J. (2006). ECSAGO: Evolutionary Clustering with Self Adaptive Genetic Operators. *Proc. of the IEEE Congress on Evolutionary Computation, 2006,* pp. 1768 – 1775.

Li, C. and Biswas, G. (2002). Unsurprised learning with mixed numeric and nominal data. *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 4, pp. 673-390.

Lubbe, J. (1997). Information theory. Cambridge University Press, Cambridge, New York.

MacQueen, J.B. (1967), Some methods for classification and analysis of multivariate observations. *Proc. of 5th Berkeley Symp. on Mathematical Statistics and Probability*, vol. 1, pp.281-297.

Matthews, G. and Hearne, J. (1991). Clustering without a metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, Feburary.

Michalski, R. (1998). Knowledge acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjunctive concepts. *International Journal of Policy Analysis and Information System*, vol. 4, pp. 210-243.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*. Spriner-Verlag Berlin Heidelberg, 3rd edition, New York.

Michaud, P. (1997). Clustering techniques. *Future generation computer system 13 (2-3)*, Elserier, Netherlands, November, pp. 135-147.

Ng, R. and Han, J. (1994). Efficient and effective clustering method for spatial data mining. *Proc. of 1994 Int. Conf. VLDB (VLDB'94)*, Santiago, Chile, Sept, pp. 144-155.

Park, K. and Carter, B. (1995). On the effectiveness of genetic search in combinatorial optimization. *Proc. of 10th ACM Symposium on Applied Computing, Genetic Algorithms and Optimization Track*, pp. 329-336.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Ramkumar, G. and Swami, A. (1998). Clustering data without distance functions. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol 21. no. 1, IEEE Computer Society, March, pp. 9-14.

Ratha, N. K., Jain, A. K. and Chung, M. J. (1995). Clustering using coarse-grained parallel genetic algorithm: a preliminary study. *Proc. of the Computer Architectures for Machine Perception*, Como, Italy, September, pp. 331-338.

Ruspini, E. R. (1969). A new approach to clustering. *Information Control*, vol. 19 pp. 22-32.

Ruspini, E. R. (1973). New experimental results in fuzzy clustering. *Information Sciences*, vol. 6, pp. 273-284.

Sarails, I., Zalzala, A. and Trinder, P (2002). A genetic rule-based data clustering toolkit. *Congress on Evolutionary Computation (CEC)*, Honolulu, USA.

Sheikholeslami, G., Chatterjee, S. and Zhang, A. (1998). WaveCluster: a multi-resolution clustering approach for very large spatial databases. *VLDB*, 1998, pp. 428-439.

Simovici D., Cristofor D. and Cristofor L. (2000). *Generalized Entropy and Projection Clustering of Categorical Data*. UMass/Boston Dept. of Math. and Comp. Sci. Technical Report 00-3, 2000

Simovici, D., Cristofor, D. and Cristofor, L. (2002). Impurity measures in databases. Acta Informatica, vol. 28 (5), 2002, pp. 307-324.

Song, W., Feng, M., Wei, S., and Shaowei, X. (1997). The hyperellipsoidal clustering using genetic algorithm. *IEEE Int. Conf. on Intelligence Processing Systems,* Oct 28-31, Beijing, China, pp. 592- 596.

Srinivas, M. and Patnaik, M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656-667.

Steinbach, M., Ertoz, L. and Kumar, V. (2001). *The challenges of clustering high dimensional data*. Technical Report.

Su, M. and Chang, H. (1998). Genetic-algorithms-based approach to self-organizing feature map and its application in cluster analysis. *IEEE World Congress on Computational Intelligence. The 1998 IEEE Int. Joint Conf. on Neural Network*, vol. 1, pp. 735-740.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. *Proc. of 3rd Int. Conf. Genetic Algorithms*, San Mateo, CA: Morgan Kaufmann, pp. 2-12.

Talavera, L. (1999). Feature selection as a preprocessing step for hierarchical clustering. *Proc. of the Sixteenth Int. Conf. on Machine Learning*, Morgan Kaufmann, Inc., pp. 389-397.

Talbi, E. and Bessière, P. (1991). A parallel genetic algorithm for the graph partitioning problem. *Proceedings of the 5th International Conference on Supercomputing*, June 1991, Cologne, Germany, pp. 312-320.

Theodoridis, S. and Kourthoumbas, K. (1999). *Pattern recognition.* Academic Press, New York.

Tibshirani, R., Walther, G. and Hastie, T. (2000). *Estimating the number of clusters in dataset via gap statistic*. Technical Report, Department of Statistics, Stanford University.

Toivonen, H. (1996). Sampling large databases for association rules. *Proc. of the 22th Int. Conf. on VLDB 96'*, Aug, 1996, pp. 134-145.

Tseng, L. and Yang, S. (1997). Genetic algorithms for clustering, feature selection and classification. *International Conference on Neural Networks*, vol.3, pp. 1612-1616.

Tzafestas, S., Saltouros, M. and Markaki, M. (1999). A tutorial overview of genetic algorithms and their applications. *Soft Computing in Systems and Control Technology*, Singapore: Word Scientific, pp. 223--300.

Wang, W., Yang, J., and Muntz, R. (1997). STING: A statistical information grid approach to spatial data mining. *Proc. of 1997 Int. Conf. of Very Large Databases (VLDB'97)*, Athens, Greece, Aug., pp. 186-195.

Whitley, D. and Kauth, J. (1988). Genitor: A different genetic algorithm. *Proc. of Rocky Mountain Conf. Artificial Intelligence*, Denver, Colorado, pp. 118-130.

Witten, I. H. and Frank, E. (1999). Weka: Practical machine learning tools and techniques with java implementations. http://www.cs.waikato.ac.nz/ml/wkea. Morgan Kaufmann.

Zaki, M. J., Parthasarathy, S., Li, W. and Ogihara, M. (1997). Evaluation of sampling for data mining of association rules. *Proc. of the seventh Int. workshop on research issues in data engineering*.

Zhang, Y., Fu, A., Cai, C., and Heng. P. (2000). Clustering categorical data. *Proc. of the 16th ICDE*, 305, San Diego, CA.

Zhang, T., Ramakrishnan, R. and Livny, M. (1996). BIRCH: An efficient data clustering method for very large databases. *Proc. of 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 96)*, Montreal, Canada, June 1996, pp. 103 –144.