Data Mining and Database Systems: Integrating Conceptual Clustering with a Relational Database Management System

Konstantina Lepinioti

Bournemouth University

A thesis submitted in partial fulfillment of the requirements of

Bournemouth University for the degree of Doctor of Philosophy

March 2011

Acknowledgements

I would like to express my appreciation to all the people who have contributed, directly or indirectly, to the completion of this thesis.

Above all, I would like to thank my supervisor, Doctor Stephen McKearney, for being a great support and inspiration all these years. I am grateful to him for engaging my interest in the area of data mining and databases. I am also grateful to him for guiding and encouraging me from the beginning until the end of this journey.

I would like to thank my second supervisor, Professor Sally McKlean, for her contribution in this work, mainly, in the early years of my research.

I would also like to thank my parents (Stergios and Eleni) and my sisters (Zoe and Ioanna) for their support and understanding, particularly, at the difficult times of anger and frustration.

I would like to express my thanks to the data mining group of the British Telecom Ipswich research centre, which funded my MPhil research that lead to this thesis.

Finally, I would like to thank Sudipto Guha, Vipin Kumar and Michael Steinbach for providing the source code for the ROCK algorithm and Periklis Andritsos for providing executables for the LIMBO algorithm.

Abstract

Many clustering algorithms have been developed and improved over the years to cater for large scale data clustering. However, much of this work has been in developing numeric based algorithms that use efficient summarisations to scale to large data sets. There is a growing need for scalable categorical clustering algorithms as, although numeric based algorithms can be adapted to categorical data, they do not always produce good results. This thesis presents a categorical conceptual clustering algorithm that can scale to large data sets using appropriate data summarisations.

Data mining is distinguished from machine learning by the use of larger data sets that are often stored in database management systems (DBMSs). Many clustering algorithms require data to be extracted from the DBMS and reformatted for input to the algorithm. This thesis presents an approach that integrates conceptual clustering with a DBMS. The presented approach makes the algorithm main memory independent and supports on-line data mining.

Contents

List of Figures

Chapter 1

Introduction

Clustering is a widely used data mining technique that has applications in areas such as marketing, to identify similar customer behaviour, and the WWW, to discover similar access patterns (Berry & Linoff, 2004; Dunham, 2003). The goal of clustering is to discover natural clusters in unclassified data that can then be studied to learn interesting rules (Everitt, 1993).

Many clustering algorithms have been developed and improved over the years for the purpose of analysing large data sets. Some of the most successful large scale clustering algorithms are numeric based algorithms that use efficient summarisations of the data. However, large data sets often contain categorical data that is not as easy to summarise.

Categorical clustering algorithms have proved more difficult to scale to large data sets (Andritsos et al., 2004; Zhang et al., 2000). One approach to achieve scalability when clustering categorical data is to use numeric based algorithms. This solution is not always successful in discovering a good clustering as it may fail to discover the natural clusters in the data (Guha et al., 2000).

An alternative approach is to investigate scaling existing categorical clustering algorithms (Andritsos, 2004). A well known categorical clustering algorithm is the conceptual clustering algorithm Cobweb (Fisher, 1987). As a conceptual clustering algorithm, Cobweb organises data into categories that maximise the similarity of data in the same category and minimise the similarity of data in different categories (Michalski, 1980). Cobweb has been successfully used in predicting missing data (Biswas et al., 1998) and personalisation of internet services (Paliouras et al., 1999). Furthermore, some of its features that work well with large data sets have been used in more recent algorithms, for example Cobweb's ability to process data sequentially (Zhang, 1997).

The main disadvantage of Cobweb as a conceptual clustering algorithm is its inability to scale to large data sets. The aim of this thesis is to investigate if it is possible to scale conceptual clustering to large data sets of categorical data.

Most of the data used in clustering exists in databases. Unfortunately, like other data mining techniques, clustering is performed, mainly, outside the DBMS using algorithms that are restricted by the available resources (Dunkel et al., 1997; Netz et al., 2000; Ordonez & Omiecinski, 2004). As a result, the overall data mining task can be complex. The user has to extract the data from its storage environment and be aware of any resource limitations (Kepner & Kim, 2003) to be able to apply the clustering algorithm and obtain a clustering. Another aim of this thesis is to investigate if it is possible to make conceptual clustering main memory independent and applicable directly to data that exists in a DBMS.

Data stored in a database is normally updated frequently creating a need for dynamic or on-line data mining. Unfortunately, many of the data mining al-

gorithms perform static data mining (Dunham, 2003). For clustering this means that once an algorithm builds a set of clusters these cannot be updated. In addition, data stored in a database is of a variety of types. Therefore, clustering algorithms that can handle different data types are more suitable. There are clustering algorithms that can handle data of different data types (mixed data) but the majority are one type algorithms, often numeric based. Also, data stored in a database is structured data following a conceptual and logical model (Date, 1995; Elmasri & Navathe, 2000). There are already examples of work in the literature that attempted to use either one or the other model to support data mining (Ketterlin et al., 1995), but most of the clustering algorithms in the literature are applicable to data that exists in a flat file form. Another aim of this thesis is to identify clustering algorithm properties that suit the characteristics of data that exists in a DBMS.

Algorithms used in a DBMS, for example indexing algorithms (Sellis et al., 1987), require minimum input from a database user. Another aim of this thesis is to identify clustering algorithm properties that suit a DBMS.

1.1 Contributions

This thesis makes the following contributions:

- Proposes CLIMIS, a scalable conceptual clustering algorithm for categorical data.
- Shows how conceptual clustering for categorical data can be scaled to large data sets by using data summarisations appropriate for categorical data.

- Shows how conceptual clustering for categorical data can be scaled to large data sets without incurring any loss in the quality of the clustering results.
- Shows how to achieve main memory independence by using a cache that integrates conceptual clustering with a relational DBMS.
- Uses an approach to clustering that supports on-line clustering.
- Uses an approach to clustering that can be extended to mixed data and has immediate database applications.
- Uses an approach to clustering that can be applied to structured data.
- Uses an approach to clustering that is possible to apply directly to data that exists in a DBMS.

1.2 Organisation of the Thesis

The organisation of this thesis is as follows. Chapter 2 presents a discussion of representative clustering techniques and algorithms. Chapter 3 is a discussion of our algorithm CLIMIS and how it scales to large data sets. Chapter 4 presents an evaluation of CLIMIS and compares it with other algorithms that also have been developed for clustering large categorical data sets. Chapter 5 presents designs that reflect the implementation of CLIMIS. Chapter 6 is a discussion of different motivations, approaches and considerations when integrating an algorithm with a DBMS. Chapter 7 discusses how we integrated CLIMIS with a relational DBMS. Chapter 8 presents an evaluation of the extended CLIMIS and the thesis ends

with chapter 9, which includes the conclusions on this work and the ways we plan to extent our work with future research.

Chapter 2

Clustering

Clustering is a research area in the fields of machine learning, statistics, databases and data mining. Clustering is employed in a number fields to infer knowledge from data. For example, clustering is used in marketing for identifying customer segments, in biology for grouping plants and animals based on their features, in telecommunications and insurance for fraud detection and in relation to the world wide web to discover groups of similar access patterns based on web log data. The problem of clustering data sets is that, quite often, the available clustering algorithms do not satisfy the clustering requirements of these fields. One of the clustering requirements is to cluster large data sets as the data collected in these fields increases every year. Dunham (2003) suggests that data doubles every year, but useful information seems to be decreasing.

This chapter starts by discussing various approaches to clustering and their problems with clustering large data sets. The chapter then presents how these approaches have been adapted to cluster large data sets.

2.1 Definition

Clustering is the grouping of a set of data into subsets, called clusters, such that the data are similar when they appear in the same cluster and dissimilar when they appear in different clusters (Everitt, 1993; Fasulo, 1999; Fraley & Raftery, 1998; Hartigan, 1975; Hinneburg & Keim, 1999). A good clustering of a given data set is one that maximises the similarity between data in the same cluster and dissimilarity between data in different clusters. A clustering algorithm normally seeks for a good trade off between the two.

2.2 Data Types

As already mentioned, earlier in this thesis, most of the data used in data mining comes from a database. Interestingly, the data mining view of data, and therefore clustering, is different to that of databases.

A database stores data as a set of relations. A relation is a set of tuples and each tuple is a set of attribute values. Each attribute value has a data type determined by the data type of the attribute.

Two major data types that are found in databases, particularly commercial databases, are numeric and categorical data. The data mining view of data is based on statistics and is more complex than the database view of data. A statistical view of numeric and categorical data, identifies numeric and categorical (also known as categoric) as two main categories of data that also include subcategories. Two subcategories of numeric are ratio and interval data and two subcategories of categorical are nominal and ordinal data (Hastie et al., 2001;

Pyle, 1999; Stevens, 1946). Each subcategory indicates important characteristics of the data.

- Numeric Data
  - *Ratio Data*: Ratio data is numbers and is not categorised. Ratio data is on a scale that has a meaningful zero and a common distance between individual points. Data that can be counted such as money and age is normally ratio data. For example, £50 is twice as much compared to £25.
  - *Interval Data*: Unlike ratio data, interval data is on a scale with no common distance between individual points. For example, the Celsius temperature scale is considered to be an interval scale because 50 degrees Celsius is not twice as hot as 25 degrees Celsius.
- Categoric Data
  - *Nominal Data*: Data is considered to be nominal when it is possible to put it into categories but these categories cannot be put into any order. Examples of nominal data are: nationality, gender and religion.
  - *Ordinal Data*: Ordinal data can be put into categories. Unlike nominal data, ordinal data categories can be put into an order. Examples of ordinal data are: -
    - rates (A, B, C, D),
    - attitudes (strongly agree, agree, disagree, strongly disagree), and
    - body mass index categories (anorexic, slim, normal, overweight, obese)

This statistical view of data is important for data mining as each data type may require different analysis and have different limitations when it comes to large data analysis. For example, the analysis of nominal data is considered more complex because it happens at attribute value level, whereas the analysis of numeric data happens at attribute level.

2.3 Similarity and Dissimilarity

A clustering algorithm makes use of a similarity or dissimilarity measure to quantify the quality of a clustering. The terms similarity and dissimilarity are often used interchangeably (Han & Kamber, 2000; Milligan, 1996). This is because, in the literature, particularly with numeric data, dissimilarity is viewed as another aspect of similarity (Gowda & Diday, 1992). In general, a similarity measure shows the strength of the relationship between two objects. A higher measurement indicates a greater similarity between two objects. A dissimilarity measure shows the distance between two objects. A lower measurement indicates a lesser dissimilarity between two objects.

The suitability of a measure for the analysis of a data set depends on the type of the data. For example, distance based measures are particularly suitable for clustering numeric data as numeric data can be easily viewed as points in a metric space, metric data.

2.4 Clustering Techniques

Clustering algorithms follow different clustering techniques. This section discusses different clustering techniques and examples of algorithms that follow these techniques.

2.4.1 Partitional and Hierarchical Clustering

A clustering algorithm can be described as partitional or hierarchical depending on the approach it follows to produce a set of clusters (Jain & Dubes, 1988).

Partitional Clustering

Given a database D of n tuples and an input number k that represents the number of desired clusters, a partitional algorithm applies a measure and partitions the data into a flat arrangement of k clusters.

A well known partitional algorithm is the k-means algorithm (MacQueen, 1967). The k-means algorithm is a distance based algorithm that views data as points in a metric space (discussed in detail in section 2.4.2). Furthermore, k-means represents a cluster by a centroid. A definition of a centroid is given by Al-Harbi & Rayward-Smith (2006) as follows:

Definition 1 *Given any set of points, C, in a metric space, M, with metric, d, a point $\hat{c} \in M$ is called a centroid if*

$$\sum_{x \in C} d(\hat{c}, x) \quad \textit{is minimised,} \tag{2.1}$$

where $d(\hat{c}, x)$ represents the distance between two points. Note that the centroid $\hat{c}$ is not necessarily an element of C.

There are different variations and extensions of the k-means algorithm (Huang, 1998; Kanungo et al., 2002; Pelleg & Moore, 2000). The original k-means algorithm that other algorithms have been based on follows the steps shown below:

```
1. Randomly select k initial centroids to represent k clusters.
2. Assign all tuples to their closest centroid using
 a similarity/dissimilarity measure.
3. Recalculate the centroids of the clusters.
4. Reassign the tuples according to the new centroids.
5. Repeat 3 and 4 until the same tuples are assigned
 to the same clusters in consecutive iterations.
```

Han & Kamber (2000) give the complexity of k-means as $O(nkt)$, where n is the number of tuples to be clustered, k is the number of clusters desired and t is the number of iterations. To achieve the same clusters in consecutive iterations and produce a set of clusters, the algorithm may require a high number of iterations.

Another partitional algorithm, similar to k-means, is the k-medoids algorithm, which is also known as PAM (Partitioning Around Medoids) (Kaufman & Rousseuw, 1990). The k-medoids algorithm follows similar steps to k-means. It looks for a good set of clusters through iterations and at every iteration it resets the medoids and reassigns the points to the best clusters. The main difference between the two algorithms is that k-means uses a mean to represent the centre of

the cluster, whereas k-medoids uses a medoid. A medoid is an object that ideally is the most central object in the cluster: A point $\hat{m} \in M$ is called a medoid if it minimises the objective function, $\sum_{x \in C} d(\hat{m}, x)$ (Mirkin, 2005). The medoid must itself be an element of C.

An advantage of the k-medoids algorithms is that it is less sensitive to any noise or outliers than the k-means algorithm (Al-Zoubi, 2009; Dehuri et al., 2006; Han & Kamber, 2000; Zhang & Couloigner, 2005). The reason is that a medoid is less affected by an outlier or other extreme values than a mean. A disadvantage of the k-medoids algorithm is that it is costly and only applicable to small data sets (Han & Kamber, 2000).

Part of the research on partitional algorithms has focused on scaling partitional algorithms to large data sets. An example of a partitional algorithm designed to cluster large data sets is CLARA. CLARA is based on k-medoids and uses a sampling based method to deal with large data sets (Kaufman & Rousseuw, 1990). It draws a random sample from the original data set and then applies the k-medoids algorithm. To produce a better clustering partition, the algorithm applies the same process a number of times. According to Han & Kamber (2000), CLARA's effectiveness depends upon the size of the sample.

CLARANS is a more recent algorithm also based on k-medoids and sampling (Ng & Han, 1994, 2002). Unlike CLARA, they use a dynamic approach to sampling. The algorithm starts with an original clustering partition based on a random sample that they call the *current clustering*. When the algorithm replaces a single medoid with a medoid randomly drawn from the data set, the new clustering partition, called *neighbour*, is compared to the *current partition* and if it is better it replaces the *current clustering*. The algorithm tries a number

of *neighbours* depending on an input parameter decided by the user. CLARANS, according to Han & Kamber (2000), shows better quality than CLARA, but worst complexity.

Hierarchical Clustering

Hierarchical clustering is different from the partitional approach in that, traditionally, it requires no input clusters and builds clusters at different levels of a hierarchy. The output of the hierarchical approach is a dendrogram (tree) that includes clusters at different levels of generalisation or specialisation.

A hierarchical algorithm can follow an agglomerative or divisive approach to clustering.

2.4.1.1 Agglomerative Clustering

Agglomerative clustering follows a bottom up approach to clustering (Duda & Hart, 1973). It starts with singleton clusters. If the data set contains n tuples, the algorithm starts with n clusters. The algorithm gradually merges the most similar pair of clusters to form a larger cluster until all data is covered by a universal single cluster. An example of an agglomerative algorithm is AGNES (Han & Kamber, 2000; Kaufman & Rousseuw, 1990). AGNES starts with singleton clusters and creates a dendrogram (tree) of clusters by gradually merging clusters whenever the similarity between them is greater than a given threshold.

2.4.1.2 Divisive Clustering

A divisive clustering algorithm starts with one cluster that covers all the cases. The algorithm splits clusters into smaller ones until every cluster is a singleton and

covers only one tuple. An example of an early divisive algorithm is DIANA (Han & Kamber, 2000; Kaufman & Rousseuw, 1990). The algorithm starts with a single cluster that covers all the data and gradually splits it into smaller clusters by applying a measurement function that indicates the furthest nearest-neighbours.

There are algorithms that have combined the partitional and hierarchical approach in order to achieve applicability to larger data sets. For example, the original agglomerative approach started with every point being a cluster on its own. The evaluation process that indicated which clusters to merge was an expensive process. As a result, more recent algorithms that follow the agglomerative approach start with k number of clusters, like the partitional approach, instead of n clusters (Andritsos, 2004).

ITERATE (Biswas et al., 1998) is another example of an algorithm that combines the hierarchical and partitional approach. ITERATE starts with a divisive algorithm that produces a clustering tree. Based on a measure of clustering quality, a set of clusters are picked from a level of the tree and they are used as an input to a partitional phase of the algorithm to improve the clustering quality.

The sections that follow discuss hierarchical algorithms or algorithms that combine the partitional and hierarchical approach as they have been more successful at clustering large data sets.

2.4.2 Distance Based Clustering

A large number of algorithms, including k-means, are described as distance based clustering because they view data as points in a metric space. The similarity or dissimilarity between two points $\hat{x}$ and $\hat{y}$, both in a metric space M, is evalu-

ated by measuring the distance between them, $d(\hat{x}, \hat{y})$. To measure the distance between points, measures that are known as *metrics* are used.

Given three data points $\hat{x}, \hat{y}$ and $\hat{z}$ all in M, a distance metric should satisfy the following (Han & Kamber, 2000; Kaufman & Rousseuw, 1990):

1. $d(\hat{x}, \hat{y}) \geq 0$
2. $d(\hat{x}, \hat{y}) = 0$ if and only if $\hat{x} = \hat{y}$
3. $d(\hat{x}, \hat{y}) = d(\hat{y}, \hat{x})$
4. $d(\hat{x}, \hat{z}) \leq d(\hat{x}, \hat{y}) + d(\hat{y}, \hat{z})$

The first condition says that distances are non-negative numbers. The second condition says that the distance of a point to itself is 0. Number 3 in the list is an axiom that reflects the symmetry of the distance function. Number 4 is known as *triangle inequality* and reflects the fact that the direct distance from a point $\hat{x}$ to a point $\hat{z}$ is always equal or shorter to the distance that includes point $\hat{y}$.

Two well known and widely used distance metrics that satisfy the above requirements are the *Euclidean distance* (equation 2.2) and the *Manhattan distance* (equation 2.3).

Euclidean Distance

$$d(\hat{x}, \hat{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{2.2}$$

Manhattan Distance

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{m} |x_i - y_i| \tag{2.3}$$

The above equations (equations 2.2 and 2.3) can be used for numeric (real valued) data. However, when the data is of the type categorical (nominal) then:

$$d(\hat{x}, \hat{y}) = \begin{cases} 0 & (\hat{x} = \hat{y}) \\ 1 & (\hat{x} \neq \hat{y}) \end{cases} \tag{2.4}$$

Given a database D with m attributes, $A_1, A_2, \ldots, A_m$, of the type real valued and categorical (nominal) data, i.e. mixed data, the following extended *Euclidean metric* can be used instead (Nguyen & Rayward-Smith, 2008):

$$d(\hat{x}, \hat{y}) = \sqrt{d_1^2(\hat{x}_1, \hat{y}_1) + d_2^2(\hat{x}_2, \hat{y}_2) + \ldots + d_n^2(\hat{x}_m, \hat{y}_m)}, \tag{2.5}$$

where $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_m)$, $\hat{y} = (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_m)$ and $d_i(\hat{x}_i, \hat{y}_i)$ is $|\hat{x}_i - \hat{y}_i|$ if $\hat{x}_i, \hat{y}_i \in R$, otherwise is the 0/1 metric. Generally, with mixed data, it is important to ensure that no attribute m dominates another with respect to the metric being used. Thus if, for example, one real valued attribute has values in the range $0, \ldots, 1,000,000$ then the differences in values could be substantial and other attribute values may be insignificant. Thus scaling of data prior to clustering is necessary.

In light of the extended *Euclidean metric*, an object $\hat{c}$ is called a centroid if it minimises $\sum_{x \in C} d^2(\hat{c}, x)$. The centroid $\hat{c}$ is not necessarily an element of C.

2.4.3 Categorical Data Clustering

In the literature, often, the term categorical is used to refer to nominal data (Cristofor & Simovici, 2002; Ganti et al., 1999). However, as discussed in section 2.2, there is also ordinal categorical data, which is different to nominal data and requires different analysis. For instance, the 0/1 metric (see equation 2.4) may be suitable for nominal and ordinal data but with ordinal data it ignores the ordering. Ordinal data can be encoded into real values (Rayward-Smith, 2007) so that the absolute difference between two encodings represents the distance between the corresponding two ordinal data points (see equation 2.5). This thesis focuses on the nominal categorical data type. In this section, as well as in the sections and chapters that follow this section, the term categorical will refer to the nominal data type only.

Most of the research in clustering has focused on the analysis of numeric data. Less attention has been given to the analysis of categorical data. To make the analysis of categorical data possible, often, the data is transformed to some numeric representation as many of the existing clustering algorithms can deal with numeric data only (Han, 1998).

A common transformation approach is to map the categorical data to binary (also referred to as Boolean) data and analyse it as data that can be ordered (metric data) (Pyle, 1999). A binary variable has only two states 0 and 1. The variable 0 indicates that the variable is absent and 1 that it is present (Han & Kamber, 2000; Li, 2005). For example, in figure 2.1, for attribute *Scorsese*, 1 indicates that *Scorsese* is present in $t1$ and 0 indicates that *Scorsese* is absent from $t2$. An example of a mapping of categorical data to binary (Boolean) is

	director	actor	genre
t1	Scorsese	De Niro	Crime
t2	Coppola	De Niro	Crime
t3	Hitchcock	Stewart	Thriller
t4	Hitchcock	Grant	Thriller
t5	Koster	Grant	Comedy
t6	Koster	Stewart	Comedy

	Scorsese	Coppola	Hitchcock	Koster	De Niro	Stewart	Grant	Crime	Thriller	Comedy
t1	1	0	0	0	1	0	0	1	0	0
t2	0	1	0	0	1	0	0	1	0	0
t3	0	0	1	0	0	1	0	0	1	0
t4	0	0	1	0	0	0	1	0	1	0
t5	0	0	0	1	0	0	1	0	0	1
t6	0	0	0	1	0	1	0	0	0	1

Figure 2.1: Mapping Categorical Data to Boolean

shown in figure 2.1. As the figure shows a Boolean attribute is created for every value in every attribute domain.

Categorical data analysis based on a Boolean mapping is a common approach but has the following disadvantages:

- It adds complexity to the KDD process as it involves more data preparation.
- It increases the number of fields.
- It requires knowledge of the entire domain of every attribute before the data mining task starts and, therefore, data mining cannot happen in an ad hoc manner (Chaudhuri et al., 1999).
- It may fail to discover the natural clusters hidden in the data or produce clusters that are not meaningful (Huang, 1997; Li & Biswas, 2002).

Furthermore, Guha et al. (2000) gives the following example to demonstrate that the use of distance metrics may be inappropriate for categorical data:

Example 1: Consider a market basket database containing the following transactions over items 1, 2, 3, 4, 5 and 6:

1. {1, 2, 3, 5}
2. {2, 3, 4, 5}
3. {1, 4}
4. {6}

A mapping of the transactions to Boolean (0/1) attributes for every item (1, 2, 3, 4, 5 and 6) transforms them and the transactions can be viewed as points:

1. (1, 1, 1, 0, 1, 0)
2. (0, 1, 1, 1, 1, 0)
3. (1, 0, 0, 1, 0, 0)
4. (0, 0, 0, 0, 0, 1)

If the transactions above are viewed as points/clusters in a metric space, following a centroid based agglomerative hierarchical algorithm (Jain & Dubes, 1988), we can apply the *Euclidean distance* (equation 2.2) and find that points 1 and 2 are the closest points: $\sqrt{(1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2)} = \sqrt{2}$. Points 1 and 2 are merged into one cluster with centroid $(0.5, 1, 1, 0.5, 1, 0)$. Points 3 and 4 are also merged as they are closer to each other than to the cluster that includes points 1 and 2. The *Euclidean distance* between 3 and 4 is $\sqrt{3}$, whereas their distance from the cluster with the merged points is $\sqrt{4}$ in both cases. The

problem in this case is that points 3 and 4 correspond to the transactions $\{1, 4\}$ and $\{6\}$ respectively, which have nothing in common.

As the example above demonstrates, categorical data cannot always be clustered properly by transforming it to Boolean and analysing it as metric data. Nguyen & Rayward-Smith (2008) recommended using the extended *Euclidean metric* (see equation 2.5) for categorical and mixed data. However, even with a more suitable metric, the approach still suffers the Boolean mapping disadvantages listed above.

The research interest on categorical data algorithms has been increasing over the years so categorical data can be clustered without having to be transformed (Barbará et al., 2002; Chen & Liu, 2005; Cristofor & Simovici, 2002; Gibson et al., 2000; He et al., 2006). Section 2.6 gives a detailed discussion of some examples of categorical data algorithms.

Mixed Data Clustering: As categorical data clustering is gradually attracting more research interest, mixed data clustering, where categorical and numeric data can be clustered together, becomes more feasible (Li & Biswas, 2002).

There are different approaches to mixed data clustering. One of the approaches involves clustering the categorical and numeric data separately using suitable measurement functions. The resulting clusters are then combined to produce mixed data clusters (He et al., 2005). Another approach to handling mixed data clustering is clustering the categorical and numeric data together but evaluating it separately. To decide on a clustering strategy, the algorithm merges the results of the numeric data evaluation and the categorical data evaluation. An example of an algorithm that follows this approach is the k-prototypes algorithm

(Huang, 1997). To evaluate the similarity between two tuples of mixed data, k-prototypes uses the k-means algorithm for evaluating the numeric data and the k-modes algorithm for evaluating the categorical data (the k-modes algorithm is discussed in section 2.6.1): if s_n is the similarity measure of numeric data and s_c is the similarity measure of categorical data then the mixed data similarity is given by: $s_n + \gamma s_c$. Huang (1997) uses γ, a user defined parameter, to balance the two parts.

A similar approach to that of Huang (1997) is used in Cobweb/3 (McKusick & Thompson, 1990). The Cobweb/3 approach is based on Cobweb and its extension CLASSIT (Gennari et al., 1989). Cobweb/3 evaluates the similarity between tuples using the *category utility (CU)* measurement (discussed in chapter 3, section 3.1). The algorithm calculates CU for categorical data and CU for numeric data separately and then it combines the two to calculate the similarity between tuples of mixed data: $CU_{mixed} = CU_{categorical} + CU_{numeric}$.

2.4.4 Other Clustering Techniques

Other clustering techniques that will only be discussed briefly in this thesis include the following:-

- Grid-based clustering
- Density based clustering
- Fuzzy clustering
- Semi-supervised clustering

Grid-based clustering views data as points in space but its difference to distance based clustering is that every dimension of the space is divided into equal intervals to discover subspace clusters (Wang et al., 1997). Grid-based clustering is mainly used with spatial data, which is 2D or 3D data, normally found in geographic information systems.

Density based clustering (Ester et al., 1996) has been developed with spatial data in mind. It uses the concept of density, which is defined as a minimum number of points within a certain distance from each other. It aims to discover clusters with the maximum of density connected points.

In conventional clustering, every tuple belongs to one cluster. Clusters produced in a conventional way, however, may not be well separated. In that case, fuzzy clustering, which allows a tuple to be a member of more that one cluster, is more appropriate (Jain & Dubes, 1988; Kaufman & Rousseuw, 1990). In fuzzy clustering, a tuple is assigned to a cluster with a degree of cluster membership.

Semi-supervised clustering happens when the data in a data set is not entirely or accurately labeled (Bouchachia & Pedrycz, 2006). Supervised learning algorithms (Liu et al., 2002; Martin, 1994) require an output field to guide the analysis process. This output field labels the tuples and any evaluation to classify the tuples is done with regard to the output field. Unsupervised learning algorithms (Iba & Langley, 2001; Oates, 2002; Song et al., 2003; Surdeanu et al., 2005), on the other hand, do not require an output field to guide their analysis. When the two learning approaches are combined, the learning is referred to as semi-supervised learning (Basu et al., 2006; Kumar et al., 2005).

2.5 Large Scale Clustering

Scalability is one of the main research areas in clustering. There are a number of approaches used to scale clustering algorithms to large data sets:

- Scalability through summarisation
- Scalability through compression
- Scalability through parallelisation
- Scalability through sampling
- Scalability through data partitioning

Scalability Through Summarisation

One approach used to scale existing algorithms to large data sets makes use of data structures that summarise the data set (Serazi et al., 2004). This approach is based on the idea that if sufficient statistics about the clusters are made available, then, the entire clusters are not needed in main memory.

Conventional distance based clustering algorithms, for instance k-means, represent a cluster with the data points that the cluster covers. Unlike those conventional distance based algorithms, there are more recent distance based algorithms that summarise the cluster representation and reduce it to the following sufficient statistics: i) a number that shows how many data points the cluster covers, ii) the sum of the data points that the cluster covers, and iii) the sum of the squared data points that the cluster covers. The work presented by Bradley et al. (1998) falls under this category. They refer to the group of the data points

to be summarised with the term *sub-cluster* and represent the *sub-cluster* with statistics that are adequate for computing the metric measure the algorithm uses (Bradley et al., 1998):

Definition 2 *Let* $\{x^1, x^2, \ldots, x^N\} \subset R^n$ *be a set of singleton points to be compressed. The statistics that represent a sub-cluster are the triple* $(SUM, SUMSQ, N)$*, where* $SUM := \sum_{i=1}^{N} x^i \in R^n$ *and* $(SUMSQ)_j := \sum_{i=1}^{N} (x_j^i)^2$ *for* $j = 1, 2, \ldots, d$.

A similar cluster representation is used by Zhang (1997). Zhang (1997) introduces the BIRCH algorithm. BIRCH relies on metrics such as the *Euclidean distance* to evaluate the clusters and decide on a clustering strategy. BIRCH represents each cluster as a *clustering feature (CF)* and provides in this way adequate statistics for the calculation of the metric measure (note that Zhang (1997) views each data point as a d-dimensional vector):

Definition 3 *Given* N *d-dimensional data points in a cluster* $\{\vec{X}_i\}$*, where* $i = 1, 2, 3, \ldots, N$*, the Clustering Feature (CF) entry of the cluster is defined as a triple:* $CF = (N, \vec{LS}, SS)$*, where* N *is the number of data points in the cluster,* $\vec{LS}$ *is the linear sum of the* N *data points, i.e.* $\sum_{i=1}^{N} \vec{X}_i$ *and* SS *is the square sum of the* N *data points, i.e.* $\sum_{i=1}^{N} \vec{X}_i^{\,2}$.

Both cluster summarisations, the *clustering feature* and the cluster summarisation of Bradley et al. (1998), use a number, N, to represents the number of points in a cluster. Also, both cluster summarisations have a similar way to represent the sum of the data points in a cluster, $\vec{LS}$ of the *clustering feature* is equivalent to the SUM of the cluster summarisation of Bradley et al. (1998). The two cluster summarisations differ with regard to SS and $SUMSQ$. Zhang (1997)

clearly states in her thesis that she views each data point as a d-dimensional vector (Zhang, 1997) and that she makes use of the vector dot product operation. With the application of the vector dot product, SS is represented by a number and is more compact than the $SUMSQ$ of Bradley et al. (1998) [1].

Scalability Through Compression

Both of the works that have been discussed above in relation to the summarisation approach (Bradley et al., 1998; Zhang et al., 1996) made use of additional techniques to compress the data further and overcome the problem of limited resources.

Bradley et al. (1998) introduced a system that is based on the idea that in the data there are regions that are compressible, regions that must be available in main memory and regions that can be discarded. The amount of compression required in their system is determined by the size of the specified memory buffer in the system. In a two phase compression approach, the system first discards from the buffer points that appear unlikely to change cluster membership and then in the second phase they compress the data to produce the cluster summarisations described in the previous section. Unfortunately, Bradley et al. (1998) fail to show any figures that clearly demonstrate the scalability of their algorithm.

The BIRCH algorithm (Zhang et al., 1996) makes use of the *clustering feature* (discussed in the previous section) and builds a tree of clusters, the *CF-tree*. A *CF-tree* is a height balanced tree with a user defined parameter called the branching factor (B for non-leaf nodes and L for leaf nodes) that controls

[1] $SUMSQ$ is represented with d numbers, where d is the number of dimensions (attributes) in the data set, whereas SS is represented with a single number regardless the dimensions in the data set.

the number of entries in a node. A non-leaf node can fit at most B entries and a leaf node can fit at most L entries. For example, in figure 2.2 $B = L = 3$ and a node (non-leaf or leaf) can fit at most 3 entries. Every entry in a leaf node is a *clustering feature (CF)*. Every entry in a non-leaf node is a summarisation of its children. Furthermore, a non-leaf node contains pointers to its children and a leaf node contains pointers to the next and previous leaf nodes to support efficient scans.

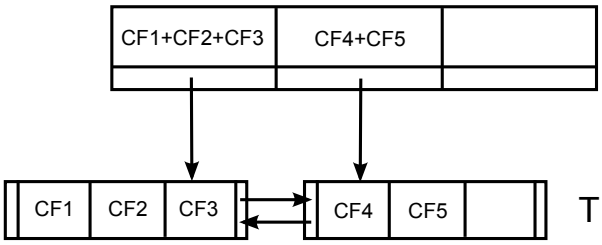

Figure 2.2: A CF-tree Example (Zhang, 1997)

To incorporate new data into the tree (new CF entry that represents one data point or is the summary of many data points), the algorithm starts from the root and using a metric measure gradually descends to the leaf node that contains the closest CF. The algorithm checks if it can incorporate the new data into the existing CF (sub-cluster). At this point, the algorithm uses a threshold T value. If the diameter of the sub-cluster (the sub-clusters are assumed to be spherical) after incorporation is less than the threshold value, the algorithm proceeds to

update the higher levels of the tree. If the diameter of the sub-cluster after incorporation exceeds the threshold value and there is a free space in the current leaf node, the algorithm stores the new CF entry separately. If there is no free space, the algorithm creates a new leaf node by performing a split of the current leaf node. To perform a split, the algorithm finds the furthest pair of CFs in the current leaf node and uses them as seeds to rearranges the rest of the CFs.

The threshold T affects the size of the *CF-tree*. The larger the threshold value the smaller the tree as more new CF entries can be incorporated into existing CFs and the algorithm has to perform less splits.

BIRCH achieves large scale clustering with limited resources by compressing the size of the *CF-tree* by using a larger threshold value when BIRCH runs out of main memory space. After the threshold value is changed, a new *CF-tree* is built based on the old one. Starting from left to right, BIRCH reads one path at a time from the old tree and creates the new smaller tree. While it creates the new smaller tree, it gradually discards the old tree as it gets replaced.

The approach of Zhang et al. (1996) has its disadvantages. BIRCH performs a node split when there is no free space in the current node. As a node split is not caused by the clustering properties of the data, it is possible a node split to have a negative impact on the quality of the clustering. To correct any negative impact on the clustering quality, the algorithm evaluates a possible merge after a split. Unfortunately, a merge may cause a new split as a merge happens in the same way an incorporation happens (described above).

The use of the threshold T may have an additional disadvantage. As long as the clusters in a data set are of spherical type and not larger in diameter than the threshold T allows, the threshold is not a problem. However, in other cases,

it may stop 'natural' clusters being discovered. New data can be incorporated into a sub-cluster only if the sub-cluster diameter after incorporation does not exceeds the threshold value (Sheikholeslami et al., 1998).

Scalability Through Parallelisation

There are algorithms that achieve scalability through parallelisation (Goil et al., 1999; Nittel et al., 2004; Skillicorn, 1999; Srivastava et al., 1999). MAFIA (Goil et al., 1999) is an example of an algorithm that uses parallelisation to produce clustering concepts on high dimensional data. MAFIA was compared with CLIQUE (Agrawal et al., 1998), another algorithm for high dimensional data sets that is a simpler sequential algorithm, and proved to be 40 to 50 times faster while producing clusters of better quality. The disadvantage of a parallel solution like MAFIA is that it is an expensive solution to implement compared to other solutions (Liu et al., 2008; Wu et al., 2005).

Scalability through parallelisation can be achieved through integration with the DBMS (an example is discussed in chapter 6, section 6.3.3).

Scalability Through Sampling

Another solution to dealing with the scalability problem is sampling. When sampling is used, the amount of data to be processed is first reduced by sampling the large data set and analysing the smaller sample. There are a number of sampling techniques that can be used to obtain a sample (Han & Kamber, 2000):

- Simple random sample without replacement: all tuples are equally likely to be drawn.

- Simple random sample with replacement: any tuple that is drawn for the sample is included in the data set and can be redrawn.

- 'Cluster' sample: the term 'cluster' here has a more general meaning. For example, a DBMS retrieves tuples in units of blocks [1]. The blocks can be considered as 'clusters' and instead of randomly drawing tuples the sampling approach can be based on randomly drawing blocks.

- Stratified sample: the data set is divided into mutually exclusive sections called *strata* and a sample is drawn from every *strata* to create the overall sample. The method is used to create representative samples when the data is skewed.

ROCK (Guha et al., 2000) is an example of an algorithm that uses sampling to scale to large data sets. The use of sampling gives ROCK the ability to cluster large categorical data sets. A disadvantage of ROCK's approach is that it may not be possible to have a small enough sample that is representative of the population. With larger data sets a representative sample may be too large to fit in main memory while a smaller sample may produce a misleading result (Lee, 1993). There are algorithms like CLARA (Kaufman & Rousseuw, 1990), for example, where the quality of the clusters returned depends upon the size of the sample (Han & Kamber, 2000). Another possible disadvantage of sampling is the objection that the owner of the data set may have about using it. Especially, in cases where data have been acquired from valuable recourses at a considerable expense, customers may insist on using the entire data set in the analysis (Wang

[1] Block is the amount of data transfered between secondary storage and main memory in a single secondary storage access (Date, 1995).

et al., 1998). Despite the possible disadvantages of sampling, it is a common approach and often the only available option for achieving scalability (Barbará et al., 2002; Guha et al., 2000).

Scalability Through Data Partitioning

An alternative to sampling when using memory dependent algorithms was proposed by Chan (Chan & Stolfo, 1993). The solution involved partitioning the data set into smaller sets so every subset fits in main memory and merging the result of each partition to draw an overall conclusion. The weakness of this approach is that the aggregation of the partitions may not be as good as the output of the whole data set.

Data partitioning, as well as the other approaches used to scale clustering algorithms (discussed earlier in this section), may have the disadvantage of limited main memory resources. The limited main memory resources may have an impact on the quality of the clusters or the algorithm may run out of main memory. A better approach is one that is main memory independent. In this thesis, we look at the problem of scalability by following the summarisation approach. Unlike other algorithms that follow the summarisation approach, our algorithm is main memory independent.

2.6 Large Scale Clustering of Categorical Data

Scaling categorical data clustering algorithms has received less attention in the literature. In this section, we discuss some representative and recent categorical data clustering algorithms that can scale to large data sets. We give particular

emphasis to the ROCK and LIMBO algorithms as we use them in chapter 4 for the evaluation of our algorithm.

2.6.1 K-modes

A well known categorical data clustering algorithm is k-modes (Huang, 1998). The k-modes algorithm extended the k-means algorithm to clustering large categorical data sets by:

1. employing a simple matching *dissimilarity measure* that allows comparison of categorical data,
2. replacing the *mean* in a cluster, used by the k-means algorithm, with the cluster's *mode*
3. using a frequency based method to update a mode in the clustering process.

Dissimilarity Measure: Let $\hat{x}$ and $\hat{y}$ denote two categorical objects described by m categorical attributes, $A_1, A_2, A_3....A_m$, the dissimilarity between the two objects can be defined as:

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{m} \delta(x_i, y_i), \tag{2.6}$$

where:

$$\delta(x_i, y_i) = \begin{cases} 0 & (x_i = y_i) \\ 1 & (x_i \neq y_i) \end{cases} \tag{2.7}$$

The smaller the number of mismatches between the two objects the more similar they are considered.

Mode of a Set: Huang (1997, 1998) defines mode in the following way. Let $\hat{\mathbf{x}}$ be a set of categorical objects described by m categorical attributes, $A_1, A_2, \ldots, A_m$.

Definition 4 *A mode of* $\hat{\mathbf{x}}$ *is a vector* $Q = [q_1, q_2, \ldots, q_m]$ *that minimises*

$$D(\hat{\mathbf{x}}, Q) = \sum_{i=1}^{n} d(\hat{x_i}, Q), \tag{2.8}$$

where $\hat{\mathbf{x}} = \{\hat{x_1}, \hat{x_2}, \ldots, \hat{x_n}\}$ and d can be defined as in equation 2.6. Q is not necessarily an element of $\hat{\mathbf{x}}$.

Huang (1997, 1998) defines a way to find a mode for a set in the following theorem. Let $n_{c_{k,j}}$ be the number of objects having category $c_{k,j}$ in attribute A_j and $fr(A_j = c_{k,j}|\hat{\mathbf{x}}) = \frac{n_{c_{k,j}}}{n}$ the relative frequency of category $c_{k,j}$ in $\hat{\mathbf{x}}$.

Theorem 1 [1] *The function of* $D(\hat{\mathbf{x}}, Q)$ *is minimised iff* $fr(A_j = q_j|\hat{\mathbf{x}}) \geq fr(A_j = c_{k,j}|\hat{\mathbf{x}})$ *for* $q_j \neq c_{k,j}$ *for all* $j = 1, \ldots, m$.

The Algorithm

The k-modes algorithm produces a flat partition of clusters in a similar way to that of k-means.

```
1. Select k initial modes to represent k clusters.
2. Assign all objects to their closest mode using
 the dissimilarity measure.
3. Update the mode of the cluster after each allocation
 according to Theorem 1.
```

[1] The proof of the theorem can be found in Huang (1998).

```
4. Reassign the objects according to the new modes.
5. Repeat 3 and 4 until the same objects are assigned
 to the same clusters in consecutive iterations.
```

The original version of k-modes selected k objects [1] from the data set to represent the initial modes. The quality of the clustering produced by the algorithm depended on the initial modes and the order the objects were presented to the algorithm. To achieve better clustering, a newer version of the k-modes algorithm was developed that uses an analysis of the frequencies of the values in the attributes $A_1, A_2, A_3, ...A_m$, which indicates better initial modes (Huang, 1998).

k-modes has the same complexity as k-means: O(nkt), where n is the number of objects to be clustered, k is the number of clusters to be produced and t is the number of iterations that the algorithm will perform. Like many other clustering algorithms, the number of clusters to be returned by the algorithm is decided by the user. The number of iterations is a parameter that is also decided by the user. According to Huang (Huang, 1998), in general, k-modes is a faster algorithm than k-means because it needs less iterations to converge than k-means.

2.6.2 ROCK

ROCK (Guha et al., 2000) is another algorithm that has been introduced for analysing large categorical data sets. The algorithm can be described as hierarchical agglomerative as it starts with a number of clusters, which then merges in order to return the desired number of clusters. The algorithm is based on the

[1] Other terms used to refer to an *object* are *tuple* and *data point*.

notions of *neighbours* and *links*.

Definition 5 *Let $sim(p_q, p_r)$ be a similarity function that shows how similar a pair of points, p_q and p_r, are. The pair of points p_q and p_r are considered to be neighbours if $sim(p_q, p_r) \geq \theta$, where θ is a user provided threshold and takes a value between 0 and 1. The similarity function could be a metric* [1] *or non-metric function.*

The above definition does not determine clearly when two points should belong to the same cluster as it is possible for points that belong to different clusters to be neighbours. For that, Guha et al. (2000) introduces the notion of links and defines $link(p_q, p_r)$ as the number of common neighbours between p_q and p_r. The higher $link(p_q, p_r)$ is, the more probable that the points p_q and p_r belong to the same cluster.

Criterion Function and Goodness Measure: The clustering aim in ROCK is to maximise the sum of $link(p_q, p_r)$ for data points belonging to the same cluster and minimise the sum of $link(p_q, p_s)$ for data point pairs belonging to different clusters. Based on this, Guha et al. (2000) define the following *criterion function* for k clusters:

$$E_l = \left(\sum_{i=1}^{k} n_i \right) \left(\sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{(1+2f(\theta))}} \right), \tag{2.9}$$

where C_i represents cluster i of size n_i and p_q and p_r represent data point pairs belonging to the same cluster. $f(\theta)$ is a function that is dependent on the data

[1] When a metric function is used, ROCK normalises the results the metric function produces and it is a larger *sim* that indicates two similar points as opposed to a lower *sim*.

set as well as the desired clusters. It is usually set to $\frac{1-\theta}{1+\theta}$. The experiments in Guha et al. (2000) use $f(\theta) = \frac{1-\theta}{1+\theta}$.

The *criterion function* ensures that points with high number of links between them appear in the same cluster. To also ensure that points with few links between them appear in different clusters, it divides the total number of links in a cluster C_i by the expected number of links in C_i, $n_i^{(1+2f(\theta))}$. The higher the *criterion function* the better the clustering produced.

As the goal in ROCK is to find a clustering that maximises the *criterion function*, they introduce the *goodness measure*, which is based on the *criterion function* and determines the best pair of clusters to merge at every step of the algorithm (Guha et al., 2000).

Given two clusters C_i and C_j with $link[C_i, C_j]$ being the number of cross links between the clusters C_i and C_j defined as $\sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r)$ [1], the *goodness measure* for merging these clusters is:

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}} \tag{2.10}$$

ROCK merges the pair of clusters with maximum *goodness measure*.

ROCK uses *links* to avoid merging clusters that appear to be similar but actually share few common neighbours. While a pair of points that exist in two different clusters that are 'not so well separated' may appear to be neighbours based on a similarity/dissimilarity function, it is unlikely that these two points have a large number of common neighbours and, therefore, unlikely for ROCK to merge these clusters.

[1] $link(p_q, p_r)$ is the number of common neighbours between p_q and p_r (discussed earlier in this section).

The Algorithm

ROCK scales to large data sets by sampling the data. A sample is drawn from a data set, then the algorithm builds k clusters based on the sample and finally it assigns the rest of the data set points to the clusters.

The ROCK algorithm is a two-phase algorithm with phase 1 building clusters based on a sample drawn from a data set and phase 2 assigning the rest of the data in the original data set to the clusters:

```
PHASE 1 - BUILD A CLUSTERING ON THE SAMPLE
1. Assign each data point of the sample to a cluster.
2. Compute the links between all possible pairs of points.
3. Merge the pair of clusters with the highest goodness measure.
2. Keep merging the clusters until k clusters are produced.

PHASE 2 - ASSIGN THE REST OF THE DATA TO THE CLUSTERS BUILT
1. Extract a few points from every cluster.
2. Compute the neighbours that a data point has in a cluster
using the extracted cluster points.
3. Assign the point to the cluster with the highest number of
neighbours.
```

ROCK uses two user defined parameters in phase 1 and two user defined parameters in phase 2. In phase 1, there is the k parameter, which determines how many clusters the algorithm will produce and the threshold θ, which determines if two points are neighbours (discussed earlier in this section). In phase 2, there is

a user defined parameter that determines the number of points that are extracted from every cluster (phase 2, step 1) and the threshold θ.

The points extracted from a cluster, in phase 2, are selected randomly and represent the cluster. Let L_i denote the extracted points from a cluster i. A data point p is compared with L_i to compute the number of neighbours. The point p is assigned to the cluster with the maximum number of neighbours. At this stage, the algorithm uses θ (phase 2, step 2) to decide the number of neighbours (see definition 5). If a point p has N_i neighbours in L_i then p is assigned to the cluster with the maximum $\frac{N_i}{(|L_i|+1)^{f(\theta)}}$. Guha et al. (2000) define $(\mid L_i \mid +1)^{f(\theta)}$ as the expected number of neighbours for point p in L_i.

ROCK can scale to large data sets but its approach is not without disadvantages (Abdu & Salane, 2009; Barbará et al., 2002). It can only scale if sampling is used and its clustering success depends on the clusters the sample produces. Also, the algorithm may have to be applied a number of times as the quality of the clusters produced on the sample depends on (i) the number of clusters requested, and (ii) the value of the threshold θ.

2.6.3 LIMBO

LIMBO (scaLable InforMation BOttlneck) is a scalable hierarchical categorical data clustering algorithm introduced by Andritsos (2004). LIMBO is based on the Agglomerative Information Bottleneck (AIB) algorithm (Slonim & Tishby, 2000). According to Andritsos et al. (2004), AIB requires a number of operations with high computational complexity, $O(n^2 m^2 \log n)$, where n is the number of clustered tuples and m is the number of different values of all attributes. The

high computational complexity makes AIB unsuitable for large data sets. LIMBO improves on AIB by using summarisation, discussed earlier in this chapter (see section 2.5).

Quality Measure and Clustering Objective: To define a quality measure for categorical clustering, LIMBO uses *mutual information* (Andritsos, 2004) and considers a good clustering one where the clusters are informative of the data they contain. According to Andritsos et al. (2004) when the data is in the form of tuples of attribute values, a good clustering is one where the clusters are informative of the attribute values in the cluster. The quality measure is the *mutual information* of the clusters and the attribute values.

Definition 6 *Given a set of tuples* $\boldsymbol{T}$ *over a set* $\boldsymbol{V}$ *of attribute values, let* $\boldsymbol{C}$ *denote a clustering of the tuples in* $\boldsymbol{T}$*. For a cluster* $C \in \boldsymbol{C}$ *and an attribute value* $V \in \boldsymbol{V}$*, the mutual information* $I(V;C)$ *measures the information about the values in* $\boldsymbol{V}$ *provided by the knowledge of a cluster in* $\boldsymbol{C}$*.*

According to Andritsos et al. (2004), clustering is a summary of the data and some information is generally lost. The clustering objective of LIMBO is to minimise this *information loss*. Merging two clusters c_i and c_j in the clustering $\boldsymbol{C}$ will produce a clustering of a smaller size, $\boldsymbol{C'}$. The information that $\boldsymbol{C'}$ contains about the values in $\boldsymbol{V}$ decreases and $I(V;C') \leq I(V;C)$. The *information loss* of merging clusters c_i and c_j is given by the following equation:

$$\delta I(c_i, c_j) = I(V;C) - I(V;C') \tag{2.11}$$

If c is the new cluster after merging c_i and c_j, then c has the following properties (Andritsos et al., 2004):

$$p(c) = p(c_i) + p(c_j) \tag{2.12}$$

$$p(V|c) = \frac{p(c_i)}{p(c)} p(V|c_i) + \frac{p(c_j)}{p(c)} p(V|c_j) \tag{2.13}$$

where $p(c) = n(c)/n$: n is the number of tuples clustered and $n(c)$ is the number of tuples in c. $p(V|c)$ is the conditional probability distribution of the attribute values given the cluster c.

Information loss can also be expressed in the following way Andritsos (2004):

$$\delta I(c_i, c_j) = [p(c_i) + p(c_j)] \cdot D_{JS}[p(V|c_i), p(V|c_j)], \tag{2.14}$$

where D_{JS} is the *Jensen-Shannon (JS)* divergence. Let $p_i = p(V|c_i)$ and $p_j = p(V|c_j)$ and let $\bar{p} = \frac{p(c_i)}{p(c)} p_i + \frac{p(c_j)}{p(c)} p_j$. Then, according to Andritsos et al. (2004),

$$D_{JS}[p_i, p_j] = \frac{p(c_i)}{p(c)} D_{KL}[p_i \| \bar{p}] + \frac{p(c_j)}{p(c)} D_{KL}[p_j \| \bar{p}]^1 \tag{2.15}$$

LIMBO chooses to merge the clusters that minimise *information loss*.

The Distributional Cluster Feature: The LIMBO algorithm tries to build cluster representations that are informative of the attribute values they cover. A cluster c is represented with a *Distributional Cluster Feature*, DCF(c):

$$DCF(c) = (p(c), p(V|c)), \tag{2.16}$$

[1] $D_{KL}[p_i \| \bar{p}]$: Relative Entropy, $D_{KL}[p_j \| \bar{p}]$: Relative Entropy (Andritsos et al., 2004; Cover & Thomas, 1991).

where $p(c) = n(c)/n$: n is the number of tuples clustered and $n(c)$ is the number of tuples in c. $p(V|c)$ is the conditional probability distribution of the attribute values given the cluster c. The DCF summarisation contains enough information to calculate the *information loss* when merging two clusters c_i and c_j as shown in the equation 2.14. If cluster c is the result of merging two clusters c_i and c_j, the DCF of the new cluster is given by the equations 2.12 and 2.13. This property of the DCF is similar to the *additivity property* of the *clustering feature* in Zhang et al. (1996) and has the advantage that the merging of two clusters involves only a few computations.

The DCF Tree: The LIMBO algorithm makes use of the *distributional cluster feature* and builds a *DCF tree*. The *DCF tree* resembles a *CF-tree* proposed in Zhang et al. (1996) (see figure 2.2). The *DCF tree* has the following characteristics:

- It is a height balanced tree.
- Every entry in a leaf node is a *distributional cluster feature (DCF)*.
- Every entry in a non-leaf node is a summarisation of its children.
- Every non-leaf node points to its children.
- A leaf node contains pointers to the next and previous leaf nodes.
- Every node in the tree can fit at most B entries.

B is a user defined parameter known as *branching factor*. For example, if B equals 3 then at most 3 entries can fit in a node of the *DCF tree*.

The algorithm uses two parameters to control the size of the tree. The first parameter is ϕ. The algorithm uses ϕ as a threshold that controls the minimum information loss that is allowed when two clusters are merged. The larger the value of ϕ, the more merges the algorithm is allowed to perform. The second parameter S controls the maximum memory size available for the *DCF tree*.

The Algorithm

LIMBO builds a clustering in three phases. Phase 1 builds a tree of cluster summarisations in an approach similar to Zhang et al. (1996). Phase 2 uses the AIB algorithm to produce an improved set of clusters based on the cluster summarisations from phase 1, and phase 3 assigns the tuples to the clusters so they can be used for interpreting the clusters.

- *Phase 1 - Insert tuples into the DCF tree*: This phase summarises the tuples. Starting from the root, the algorithm traces a path down the *DCF tree*. At each non-leaf node it meets, the algorithm computes the distance between DCF(t) (the new tuple) and every DCF entry of the node finding the closest DCF entry to the DCF(t). The distance is calculated using *information loss*. The algorithm then follows the child pointer of the entry to the next level of the tree and repeats the same process until it gets to a leaf node. When at a leaf node, the algorithm finds the DCF that is closest to the DCF(t). The algorithm has to decide, then, if merging the DCF(t) with that DCF is the best strategy.

  The decision is affected by the parameter the algorithm uses: ϕ or S. If, for example, the used parameter is S, then LIMBO's steps are controlled

according to the available memory. If the algorithm finds an empty space in the leaf node, then DCF(t) is placed in the empty space. If there is no empty space and there is available memory (according to the S parameter) then the algorithm splits the leaf node into two. The way the algorithm performs the split is as follows:-

- Check the DCFs stored in the leaf node and find the DCFs that are furthest apart (using *information loss*).
- Create two new leaf nodes with the furthest apart DCFs.
- Insert the rest of the DCFs in the new leaf nodes according to the DCF to which they are closest. DCF(t) is one of those DCFs.

If there is no empty space in the leaf node and no available memory, then the algorithm performs a merge in one of the following ways, depending on which merge offers the minimum information loss: -

- Merge DCF(t) with the DCF in the leaf node to which it is closest.
- Merge two other clusters in the leaf node by merging their DCFs.

When the algorithm finishes inserting a new tuple into a leaf node, it updates the higher levels of the tree. If a new leaf node was created because of lack of empty space in an existing leaf node, the algorithm checks the leaf node's parent for empty space. If there is an empty space in the non-leaf node, the algorithm adds a new DCF entry into the non-leaf node. Otherwise, the algorithm splits the non-leaf node in the same way it splits a leaf node, described above. The same process continues upward in the tree

until the root node is updated or split itself. If the root node is split, then the tree height increases by one level.

- *Phase 2 - Clustering* AIB is applied on the DCFs at the leaf nodes. This speeds up AIB as normally AIB starts with every tuple as a singleton cluster. The algorithm gradually merges the clusters, using the *information loss* measure, until the desired number of clusters is reached. As the number of DCFs at the leaf nodes is relatively small, the AIB has significantly less operations to perform and, therefore, shows better performance.

- *Phase 3 - Associating tuples with the clusters* LIMBO performs a scan over the data set and assigns each tuple to one of the clusters produced.

2.6.4 Summary

The algorithms discussed in the section above (section 2.6) can scale to large categorical data sets but are quite complex to apply for being parametric and, some of them, multi-phase algorithms. For example, all of them require a user to have an understanding of the data in order to decide the best number of clusters that should be requested. ROCK and LIMBO use further parameters that affect the quality of the clustering produced. ROCK uses the θ parameter, and LIMBO the B and ϕ [1] parameters, which determine which clusters will be merged. Also, ROCK and LIMBO have the additional complexity, in the way they are applied, for being multi-phase algorithms with the first phase affecting the quality of the next phase.

[1] Depending on the version of the LIMBO algorithm, instead of the ϕ parameter, the algorithm may require setting of the S parameter.

Mining large databases requires algorithms that are simple to apply (Dunkel & Soparkar, 1999; Lu, 2001). We discuss conceptual clustering in the next section as conceptual clustering provides a framework for clustering, where the clustering quality does not depend on user defined parameters and and the clustering process involves a single phase.

2.7 Conceptual Clustering

Conceptual clustering is based on numerical taxonomy (Fisher & Langley, 1986) and was originally introduced by Michalski & Stepp (1983). Gennari et al. (1989) described the problem of conceptual clustering in the following way:

> Given: a sequential presentation of instances and their associated descriptions;
>
> Find: clusterings that group those instances in categories;
>
> Find: an intentional definition for each category that summarises its instances;
>
> Find: a hierarchical organisation for those categories.

As it is stated above, conceptual clustering organises instances (tuples) into categories. This makes conceptual clustering suitable for categorical data that cannot be ordered and can only be put into categories (discussed in section 2.2). A successful conceptual clustering algorithm that has been the basis for many other algorithms, for example LABYRINTH (Thompson & Langley, 1991) and ITERATE (Biswas et al., 1998), is Cobweb (Fisher, 1987). This thesis focuses on Cobweb with the aim to extend it and scale it to large data sets of categorical data that cannot be ordered.

2.7.1 Cobweb

Cobweb is a conceptual clustering algorithm developed by Fisher (1987) for the analysis of categorical data that cannot be ordered. The algorithm builds a hierarchy of clusters following the divisive approach to clustering. The goal of Cobweb, like all conceptual clustering algorithms, is to build a model that can be used for future predictions (Gennari, 1989).

Cobweb is a relatively old algorithm but since it was introduced its relevance to solving data mining problems has remained important. Biswas et al. (1998) use Cobweb for predicting missing values. Perkowitz & Etzioni (2000) discuss the suitability of Cobweb for data mining on the web, Hurst et al. (2003) and Paliouras et al. (1999) use Cobweb on the web, while Li et al. (2005) combine Cobweb with k-means (MacQueen, 1967) to present an algorithm for large scale clustering [1]. The algorithm is, also, part of a number of popular general purpose data mining tools. Two of these data mining tools are (i) Weka (Garner, 1995), which provides an implementation of Cobweb that is applicable to categorical and numeric data, and (ii) OIDM (Chen et al., 2004), which provides an implementation of Cobweb based on the original Fisher's paper.

Cobweb builds clusters with high intra-cluster similarity and inter-cluster dissimilarity so a tuple in a cluster is similar to the tuples in the same cluster and different to the tuples in other clusters. The algorithm uses *category utility* to build a clustering that maximises intra-cluster similarity and inter-cluster dissimilarity (discussed in detail in section 3.1). The computation of *category utility* relies on the probability distribution of the tuple attribute values that the clusters cover. To support the computation of *category utility*, Cobweb represents a

[1] The paper fails to show scalability results.

cluster C as a probability distribution of the tuple attribute values in the cluster.

Definition 7 *Given a set of tuples* $\boldsymbol{T}$ *over a set* $\boldsymbol{V}$ *of attribute values, let* $\boldsymbol{C}$ *denote a clustering of the tuples in* $\boldsymbol{T}$*. For a cluster* $C \in \boldsymbol{C}$ *and an attribute value* $V \in \boldsymbol{V}$*, the Cobweb cluster representation is defined by the pair:*

$$\textit{Cobweb cluster representation} = (p(C), p(V|C)), \tag{2.17}$$

where $p(C) = n(C)/n$: n is the number of tuples clustered and $n(C)$ is the number of tuples in C. $p(V|C)$ is the conditional probability distribution of the attribute values given the cluster C.

The Cobweb cluster representation is the same as the *distributional cluster feature* used by the more recent algorithm LIMBO discussed earlier in this chapter (see section 2.6.3).

The Algorithm

The Cobweb algorithm is an incremental clustering algorithm that clusters one tuple at a time in a top down manner. It starts clustering a tuple by inserting it into the root cluster of the tree (figure 2.3 is an example of a Cobweb tree). Inserting a new tuple in a cluster involves updating the probabilities the cluster covers.

The algorithm uses four operators to evaluate and improve the quality of the tree. The quality measure in Cobweb is *category utility* (see equation 3.2, section 3.1). The four operators are: (i) *incorporate*, (ii) *disjunct*, (iii) *split*, and (iv) *merge*. The *incorporate* and *disjunct* operators are used to build the tree

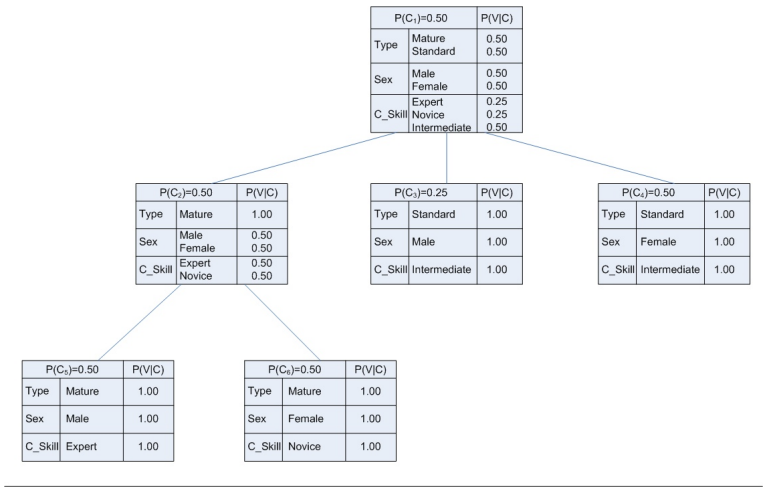

Figure 2.3: The Cobweb Tree

while the *merge* and *split* operators are used to correct any data ordering bias in the clusters by reordering the hierarchy.

- *Incorporate:* Cobweb tries a new tuple in every cluster of the assessed level to identify the best cluster to incorporate the new tuple. It also records the second best cluster as it is needed by other operators.

- *Disjunct:* Cobweb tries a new tuple in a new cluster that covers only the tuple.

- *Split:* Cobweb replaces the best cluster, identified by the *incorporate* operator, with its children and tries the new tuple in every child of the best cluster.

- *Merge:* Cobweb merges the best and second best clusters, identified by the *incorporate* operator, and tries the new tuple in the merged cluster.

According to Fisher et al. (1992), the incremental property can have an

impact on the quality of the clusters as incremental algorithms are sensitive to the order of the data (Langley, 1995). With the *merge* and *split* operators the algorithm corrects the ordering effect by restructuring the tree.

As it descends down the tree, at every level of the tree, Cobweb tries all four operators - *incorporate*, *disjunct*, *split* and *merge* - and identifies which is the *best operator* to implement by measuring the *category utility* of the clustering produced by each operator. *Category utility* favours the operator that when implemented produces a clustering that maximises the potential for inferring information (Fisher, 1987; Gennari et al., 1989) (a more detailed discussion of category utility is given in section 3.1).

```
Function Cobweb (tuple, root)
  Incorporate tuple into the root;
  If root is a leaf node Then
   Expand leaf node;
    Return expanded leaf node with the tuple;
  Else
    Get the children of the root;
    Evaluate operators and select the best:
      a) Try incorporate the tuple in every child;
      b) Try creating a new cluster with the tuple;
      c) Try merging the two best clusters;
      d) Try splitting the best cluster into its children;
    If (a) or (c) or (d) is best operator Then
      call Cobweb (tuple, best cluster);
```

If the *best operator* is *incorporate*, the algorithm inserts the new tuple in

the best cluster and proceeds to the next level. If the *best operator* is *disjunct*, the algorithm creates a new cluster in the tree. If the *best operator* is *split*, the algorithm re-arranges the tree by replacing the best cluster with its children and moves to the next level. If the *best operator* is *merge*, the algorithm merges the best and second best cluster (best and second best cluster are indicated by the *incorporate* operator) and moves to the next level. Figure 2.4 shows how the operators are implemented.

Cobweb has an additional operator used to predict missing values, the *predict* operator. The predict operator classifies a tuple down the tree using the *incorporate* operator but it does not add the tuple to the clusters in the tree.

2.7.2 Discussion

LIMBO, BIRCH and Cobweb are all incremental algorithms as they all allow new tuples to be inserted to an existing model. The incremental property is an advantage because clusters can be updated locally and updating the clusters does not involve all the previously seen tuples. ROCK, on the other hand, is not an incremental algorithm as, before it starts building the clusters it needs to compute the links between all the possible pairs of points in the sample. The k-modes algorithm, also, is not incremental. The algorithm follows the partitional approach to clustering, which involves all the tuples in the clustering process.

As hierarchical incremental algorithms, LIMBO, BIRCH and Cobweb require one scan of the data from the disc. However, LIMBO involves an additional scan of the data from the disc as after it builds a clustering it needs to associate the tuples to the clusters.

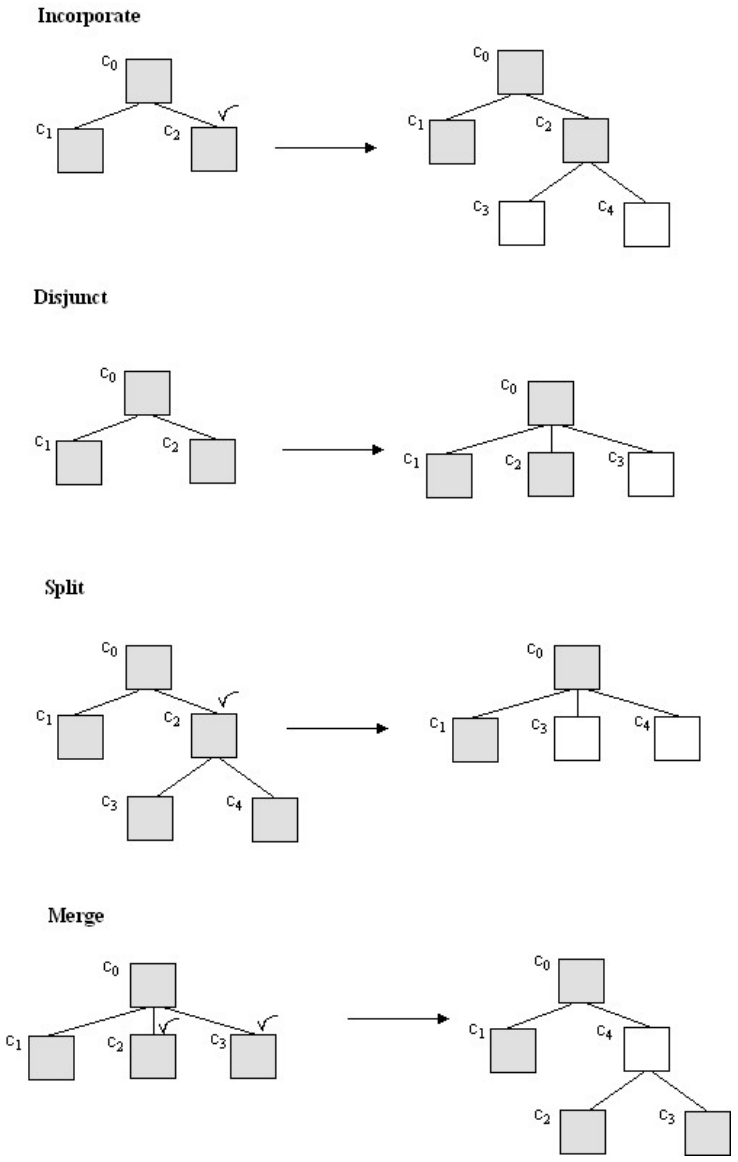

Figure 2.4: Cobweb Operators

The algorithms LIMBO, BIRCH and ROCK are all parametric algorithms. They require, for example, the user to define the number of clusters to be produced by the clustering process. The parametric characteristic of an algorithm can be a disadvantage as the user may not have knowledge of a good parameter setting with regard to a data set. In that case, the user may have to perform a large number of clustering trials to reach a good set of clusters. The only incremental algorithm that has been discussed in this chapter which is non-parametric is Cobweb (Kilander, 1993; Talavera & Roure, 1998; Theodorakis et al., 2004).

Dunham (2003) gives a database perspective of data mining and suggests that for data mining algorithms to be effective in databases, it is not enough for them to work well but they also have to be perceived by the users as easy to use and interpret. Cobweb as a non-parametric algorithm is the simplest of the incremental algorithms mentioned here. Cobweb, also, produces clusters that are simple to interpret because it produces clusters at different levels of specialisation or generalisation.

Cobweb has been implemented as a mixed data algorithm and remained incremental. The original *category utility (CU)* measurement (Fisher, 1987) used by Cobweb (defined in chapter 3, section 3.1), was extended to numeric data in CLASSIT (Gennari et al., 1989). The two versions of CU were then combined to provide a mixed data implementation. McKusick & Thompson (1990) combined $CU\_categorical$ with $CU\_numeric$ and implemented a version of Cobweb for mixed data clustering that is incremental.

From a database perspective, Cobweb is a good algorithm to use for the additional reason that it has immediate application to database problems. The algorithm offers:-

- Accurate prediction of missing values (Biswas et al., 1998). Missing values is an important problem of databases. Cobweb treats missing data as another attribute value, which is considered in the evaluation process (Fisher, 1989). According to Fisher, using the predict operator and relying on the information in the clusters, the algorithm can successfully predict the missing values.

- Ability to cluster structured data. Data stored using a DBMS follow a structure and can be described as structured data. Cobweb has already been used in clustering structured data. Two examples are the work of Ketterlin et al. (1995) and Sison & Shimura (1996). Both works researched the area of clustering database relations and are based on Cobweb.

The only database requirement that Cobweb does not support, and all the other algorithms mentioned here do, is the ability to scale. The algorithm is not able to scale because it does not use efficient data summarisations that reduce the computations it has to perform. One of the aims of this thesis is to scale Cobweb to large data sets. We focus on scaling Cobweb for categorical data. We, also, make the algorithm memory independent with the use of a cache that allows the algorithm to interact with a relational DBMS.

Chapter 3

CLIMIS Clustering

This chapter introduces CLIMIS, a scalable conceptual clustering algorithm that is based on Cobweb (Fisher, 1987). The objectives of this chapter are as follows:

1. To show a conceptual cluster summarisation that requires minimum updating and main memory space.

2. To show that the cluster summarisation proposed supports the computation of category utility.

3. To present a data structure that stores the cluster summarisation.

4. To show that the CLIMIS algorithm uses the proposed cluster summarisation and data structure to implement the same operators as Cobweb.

Section 3.1 presents *category utility*, the measure that the CLIMIS algorithm uses to build a clustering. Section 3.2 presents the CLIMIS algorithm starting with a discussion of its properties. Section 3.2.1 discusses the *cluster summarisation* CLIMIS uses to avoid detailed cluster descriptions at every level

of the clustering tree. Section 3.2.2 presents the data structure that the algorithm uses to build a conceptual clustering tree. Section 3.2.3 presents the operators that the CLIMIS algorithm implements using the proposed *cluster summarisation* and data structure. The chapter ends with a discussion of the contributions of this thesis.

3.1 Category Utility

CLIMIS uses *category utility* to evaluate a clustering. *Category utility* is a measure that has its roots in Information Theory (Shannon & Weaver, 1949). It was introduced by Gluck & Corter (1985) with the aim of predicting the basic level in human classification hierarchies. The basic level is considered to be the most natural level of categorisation, for example, dog is the basic level in the hierarchy animal-dog-poodle. Gluck & Corter (1985) suggested that certain categories or concepts, like basic level categories, are easier to learn and remember because they reduce the uncertainty about the instances they cover.

Gluck & Corter (1985) based category utility on (i) the definition of uncertainty by Shannon & Weaver (1949) and (ii) a hypothetical communication game, where a person transmits information about an item's attributes to another person. Within the hypothetical communication game, they defined uncertainty as *"..an inability to predict attributes, and analyze how category membership information can be used to transmit information about the attributes of object or events"*.

According to Fisher (1987), category utility favours clusters that maximise information inference. In doing this, category utility aims to maximise intra-

class similarity and inter-class dissimilarity and achieve the best trade-off between them. Given an attribute value pair $A_i = V_{ij}$ and a cluster C_k,

- *Intra-class similarity* is reflected by the conditional probability $P(A_i = V_{ij}|C_k)$. The larger the intra-class similarity, the greater the number of the cluster members that share $A_i = V_{ij}$ and the more predictable the attribute value is given that an instance is a member of C_k.

- *Inter-class dissimilarity* is reflected by the conditional probability $P(C_k|(A_i = V_{ij})$. The larger the inter-class dissimilarity the fewer the members of different clusters that have $A_i = V_{ij}$ and the more predictive the attribute value pair is of the cluster.

The above measures of intra-class similarity and inter-class dissimilarity of individual attribute value pairs are combined into an overall measure of clustering quality across all clusters (k), attributes (i) and values (j). The measure also includes $P(A_i = V_{ij})$, which is used to weight all values so that frequently occurring values play a more important role in the clustering quality than less occurring values (Gennari et al., 1989):

$$\sum_k \sum_i \sum_j P(A_i = V_{ij})P(C_k|A_i = V_{ij})P(A_i = V_{ij}|C_k) \tag{3.1}$$

The application of Baye's rule in 3.1 replaces $P(A_i = V_{ij})P(C_k|(A_i = V_{ij})$ with $P(C_k)P(A_i = V_{ij}|C_k)$ and produces the measure of category utility defined by Fisher (1987). Given a partition of clusters $\{C_1, C_2, ...C_l\}$

$$CU = \frac{\sum_{k=1}^{l} P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{l} \quad (3.2)$$

The denominator l in the CU equation represents the number of clusters in a partition and is used to allow comparison of different size partitions.

3.2 The CLIMIS Clustering Algorithm

CLIMIS is a hierarchical algorithm that builds clusters at different levels of summarisation making clustering easier to interpret. CLIMIS is incremental and as such it allows incorporation of new data into an existing clustering. The algorithm clusters one tuple at a time having to adjust a clustering only locally and for that it requires one scan of the data from the disk. CLIMIS normally produces a skewed tree of clusters with the leaf clusters being singletons.

The properties of CLIMIS mentioned here are also properties of the Cobweb algorithm. The main difference between the two algorithms relates to the number of operations they have to perform in order to build a clustering.

The Cobweb algorithm relies on clusters represented as discrete probability distributions. The algorithm fails to scale to large data sets (see chapter 4, figure 4.3) because the four operators it evaluates and the *category utility* measure it uses involve a number of complex operations for every tuple that is clustered.

Many of the operations that Cobweb performs are scans of the clusters to calculate CU for the *incorporate*, *disjunct*, *merge* and *split* operators. The excessive complexity of the algorithm is due to the cluster representation it uses:

$(p(C), p(V|C))$. This is the same cluster representation that LIMBO uses. However, this cluster representation does not have the same impact on LIMBO as LIMBO employs a tree reduction approach that improves the complexity of the algorithm. LIMBO reduces its complexity further by only applying two operators, split and merge (distinct from Cobweb's *split* and *merge* operators), instead of four operators.

Our approach to conceptual clustering uses a more compact cluster representation and is based on the idea that in hierarchical clustering it is not necessary to store detailed statistics at all levels of a hierarchy as long as detailed statistics are provided at the leaf node level of the tree.

3.2.1 Cluster Summarisation

We introduce the *cluster summarisation (CS)* that compresses the original Cobweb cluster representation and maintains Cobweb's important property of being incremental.

Definition 8 *Given a set of tuples* $\boldsymbol{T}$ *over a set* $\boldsymbol{V}$ *of attribute values, let* $\boldsymbol{C}$ *denote a clustering of the tuples in* $\boldsymbol{T}$*. For a cluster* $C \in \boldsymbol{C}$*, Cluster Summarisation (CS) is defined by the pair:*

$$\textit{Cluster Summarisation} = (p(C), SUMSQ), \tag{3.3}$$

where $p(C) = n(C)/n$: n is the number of tuples clustered and $n(C)$ is the number of tuples in C. $SUMSQ$ is the sum of the squared conditional probabilities of the attribute values given the cluster C: $SUMSQ = \sum_i \sum_j P(A_i = V_{ij}|C)^2$. Every cluster in the tree is represented with a *cluster summarisation*.

Leaf Cluster Representation: A leaf cluster is represented with a *cluster summarisation* (defined above) like every other cluster in the tree. Our approach uses an additional cluster representation for a leaf cluster and that is a $p(V|C)$. A $p(V|C)$ is the conditional probability distribution of the attribute values in a leaf cluster. This additional cluster representation at leaf clusters provides detailed statistics that can be used in all the levels of the tree that CLIMIS builds. A $p(V|C)$ for a non-leaf cluster can be computed from the $p(V|C)$ of the corresponding leaf clusters.

Properties of the Cluster Summarisation: Our *cluster summarisation* has the following properties:

- *Incremental:* The incorporation of a new tuple in a cluster C involves adjusting the *cluster summarisation* of C. Let T' be a new tuple to be incorporated into C. The new tuple is presented to the algorithm with the cluster representation: $(p(C'), p(V'|C'))$, where $p(C') = n(C')/n$ (n is the number of tuples clustered and $n(C')$ is the number of tuples in C'). $p(V'|C')$ is the conditional probability distribution of the attribute values in the cluster C'. To adjust the *cluster summarisation* of C, the algorithm has to perform the following:

  - $P(C) + P(C')$,
  - $\sum_i \sum_j P(A_i = V_{ij}|C)^2 + p(v'|C')^2 + 2p(v|C)p(v'|C')$,

  where $p(v'|C')$ is the conditional probability of an attribute value in C' that matches an attribute value in C. $p(v|C)$ is the conditional probability of that attribute in C. The algorithm gets $p(v'|C')$ from the new tuple's cluster representation and $p(v|C)$ from the leaf clusters.

If an attribute value in C' matches no attribute value in C then the algorithm performs the following computation:

$$- \sum_i \sum_j P(A_i = V_{ij}|C)^2 + p(v'|C')^2$$

- *Predictable:* The size of the *cluster summarisation* remains constant as the amount of data increases.

- *Compact:* Regardless of the number of attributes, tuples or the size of the attribute domains in the data set, the *cluster summarisation* is always represented by two numbers and it requires minimum main memory space.

- *Representative:* The CLIMIS *cluster summarisation* can be used by other categorical data algorithms that are based on Cobweb such as ITERATE (Biswas et al., 1991) and algorithms that are hierarchical and use a probability distribution to represent a cluster, for example LIMBO.

3.2.2 CLIMIS Tree

The CLIMIS algorithm makes use of the *cluster summarisarion (CS)* (defined in section 3.2.1) and builds a *CLIMIS tree*. A *CLIMIS tree* (see figure 3.1) has the following characteristics:

- There is one root node and every other node in the tree is one of two types: internal or leaf node.

- Every internal and leaf node in the tree can fit as many entries as required by the clustering.

- All entries in an internal or leaf node represent children of the same cluster.

- The root node has one entry of the type $[CS_i, child_i]$, where CS_i is the *cluster summarisarion* of the cluster the entry represents and $child_i$ is a pointer to the child node of the root node.

- Every entry in an internal node is of the type $[CS_i, child_i]$ if the entry represents an internal cluster or $[CS_i, p_i]$ if the entry represents a leaf cluster, where CS_i is the *cluster summarisarion* of the cluster the entry represents and p_i is a pointer to the leaf cluster's $p(V|C)$.

- Every entry in a leaf node is of the type $[CS_i, p_i]$.

- An entry of the type $[CS_i, p_i]$ that represents a leaf cluster may describe a singleton or non-singleton cluster.

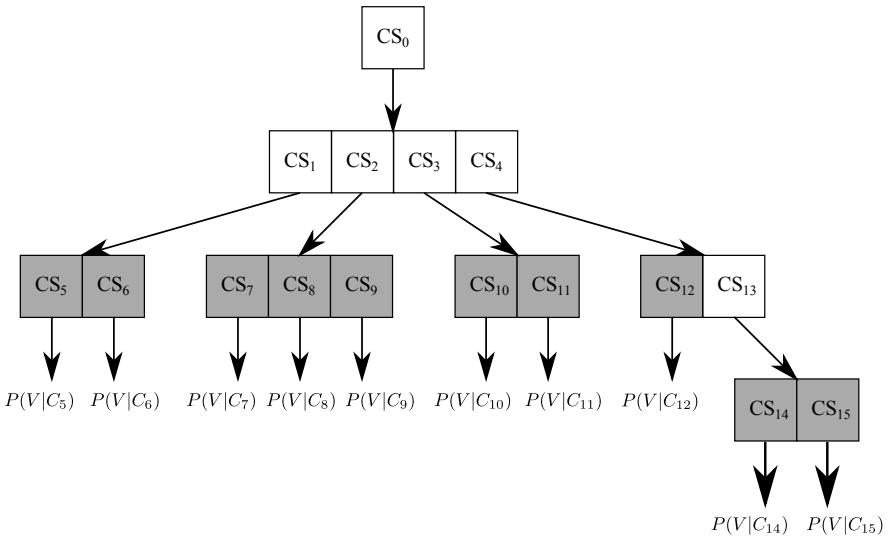

Figure 3.1: CLIMIS Tree

The PD-structure

As it has already been mentioned (page 57), our approach to conceptual clustering uses a more compact cluster representation and is based on the idea that in hierarchical clustering it is not necessary to store detailed statistics at all levels of a hierarchy as long as detailed statistics are provided at the leaf node level of the tree. This section presents the *PD-structure*, which is an auxiliary structure to the *CLIMIS tree* and provides detailed statistics at leaf cluster level.

CLIMIS uses the *PD-structure* to store additional cluster representations at leaf cluster level (see section 3.2.1). The *PD-structure* is a matrix that stores the conditional probability distribution of the attribute values, $p(V|C)$, in every leaf cluster.

p(V|C)	C_1	C_2	...	C_k
V_1	$p(V_1\|C_1)$	$p(V_1\|C_2)$		$p(V_1\|C_k)$
V_2	$p(V_2\|C_1)$	$p(V_2\|C_2)$		$p(V_2\|C_k)$
V_3	$p(V_3\|C_1)$	$p(V_3\|C_2)$		$p(V_3\|C_k)$
...				
V_j	$p(V_j\|C_1)$	$p(V_j\|C_2)$		$p(V_j\|C_k)$

Figure 3.2: PD-Structure

Let $\{V_1, V_2, V_3, \ldots, V_j\}$ denote the attribute values in $\mathbf{T}$, where $\mathbf{T}$ is a set of tuples. Also, let $\mathbf{C}$ denote a clustering of the tuples in $\mathbf{T}$ and $\{C_1, C_2, C_3, \ldots, C_k\}$ denote the clusters in $\mathbf{C}$. The *PD-structure* stores a conditional probability $p(V_j|C_k)$ of an attribute value V_j in a cluster C_k. Figure 3.2 shows the *PD-*

structure.

If a new tuple reaches a leaf cluster C_k then $p(V_j|C_k)$ can be easily updated. If $p(V_{j\prime}|C_{k\prime})$ is the conditional probability of an attribute value $V_{j\prime}$ in the new tuple that matches the attribute value V_j in C_k then $p(V_j|C_k)$ is updated with $p(V_j|C_k) + p(V_{j\prime}|C_{k\prime})$.

As it is discussed in section 3.2.1, a leaf cluster is also represented with a *cluster summarisation (CS)*. Therefore, if a new tuple reaches a leaf cluster C_k then the leaf cluster's *CS* is also updated as shown in section 3.2.1.

3.2.3 CLIMIS Algorithm

The CLIMIS algorithm implements the same operators as Cobweb:

- *Incorporate:* CLIMIS tries a new tuple in every cluster of the assessed level to identify the best cluster to incorporate the new tuple. It also records the second best cluster to be used by other operators.

- *Disjunct:* CLIMIS tries a new tuple in a new cluster that covers only this tuple.

- *Split:* CLIMIS replaces the best cluster, identified by the *incorporate* operator, with its children and tries the new tuple in every child of the best cluster.

- *Merge:* CLIMIS merges the best and second best clusters, identified by the *incorporate* operator, and tries the new tuple in the merge cluster.

As it descends down the tree, CLIMIS decides the *best operator* based on *category utility*. The operator that is indicated by *CU* to be the best is the

operator that the algorithm implements. To evaluate the operators, the algorithm needs the $p(V_j|C_k)$ of the attribute values of the clusters, in the assessed level, that match the attribute values in the new tuple. The algorithm performs one search of the *PD-structure* in order to find the attribute value probabilities that match the attribute values of a new record.

```
Function CLIMIS (tuple, root)
Get the p(Vj|Ck) that match the new tuple from PD-structure.
  Incorporate tuple into the root by adjusting the root CS;
  If root is a leaf cluster Then
    Return expanded leaf cluster with the tuple;
    Update PD-structure and add new p(Vj|Ck)
  Else
    Get the children of the root;
    Evaluate operators and select the best:
      a) Try incorporate the tuple in every child;
        Temporarily adjust every child's CS
      b) Try disjunct;
        Create a new CS with the tuple;
      c) Try merging the two best clusters;
        Create merge CS from the PD-structure;
      d) Try splitting the best cluster into its children;
        Replace best cluster CS with its children CS.
        Temporarily adjust every child's CS
    If (a) or (c) or (d) is best operator Then
     Replace best cluster CS with adjusted CS
      Call CLIMIS (tuple, best cluster);
```

The *merge* operator is the only operator that involves an additional search of the *PD-structure* to compute the merge *cluster summarisation*.

Prediction Algorithm: Like Cobweb, CLIMIS can be used to perform predictions based on a built clustering. Also like Cobweb, the prediction aspect of our algorithm uses only the *incorporate* operator. The algorithm applies the *incorporate* operator to decide the best cluster to incorporate the tuple and then moves to the next level of the tree. CLIMIS records the cluster the tuple has been incorporated into at every level of the tree. The prediction algorithm does not update the *CLIMIS tree* or the *PD-structure*.

Avoiding the Proliferation of Clusters

The Cobweb algorithm builds a tree with every leaf cluster covering a single tuple, singleton cluster. When a new tuple is incorporated into a singleton cluster, the algorithm expands the tree by making the singleton cluster an internal node and the parent of two singleton clusters, one containing the old tuple and the other containing the new tuple. As a result, when Cobweb is applied to large data sets the algorithm can return a large tree.

To avoid the proliferation of the clusters produced by Cobweb, we use the *cutoff* parameter (Witten & Frank, 2005). The *cutoff* parameter is a threshold that the algorithm uses to suppress growth. The *cutoff* is determined in terms of category utility. When the addition of a new tuple into a cluster does not improve the category utility then that cluster is cut off.

When assessing the clusters at a level of a tree, the algorithm first tries the *incorporate*, *disjunct*, *split* and *merge* operators to find the best CU (CU_{best})

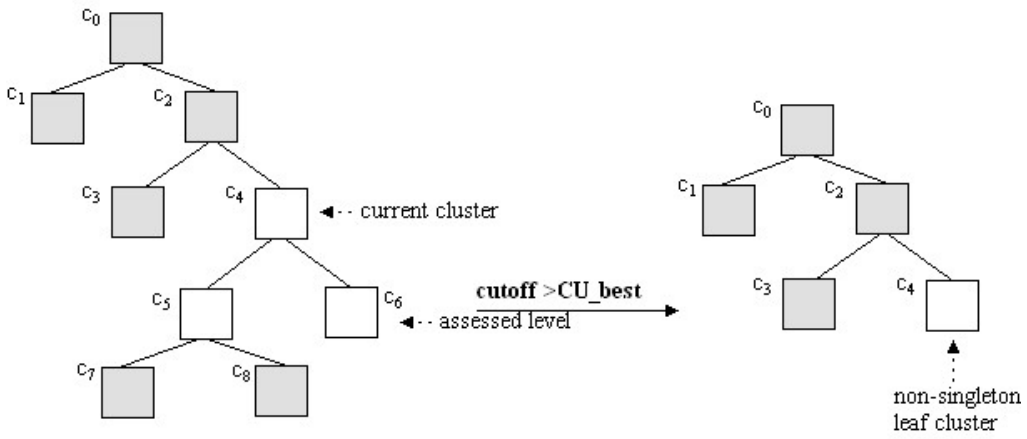

Figure 3.3: Application of the Cutoff Operator

that indicates the *best operator*. The *cutoff* is applied as an operator like the other operators of CLIMIS. Before implementing the *best operator* in the assessed level, the algorithm compares CU_{best} to the *cutoff* threshold, using the *cutoff* operator. If $CU_{best} > cutoff$ the algorithm continues as normal and implements the *best operator*. If $CU_{best} < cutoff$ the algorithm removes the assessed level and any clusters below that (see figure 3.3).

Applying the *cutoff* parameter means that the leaf cluster in that path is no longer a singleton. It also means that a number of leaf clusters have been removed from the tree. CLIMIS updates the *PD-structure* accordingly. The algorithm removes all the previous leaf clusters from the *PD-structure* and adds the new leaf cluster.

When the *cutoff* parameter is set to zero, the CLIMIS algorithm operates like Cobweb as the threshold is ignored. When CU_{best} is zero the algorithm favours CU_{best}. If the *cutoff* parameter is set to anything greater than zero, the CLIMIS algorithm tries and if appropriate implements the *cutoff*. After implementing the

cutoff operator, if the algorithm reaches a leaf cluster that is not a singleton, then the algorithm expands the leaf cluster in the same way it would expand a singleton. It makes the leaf cluster the parent of two new leaf clusters.

According to Fisher (1987), Cobweb seeks clustering trees in which the first level is optimal with respect to a measure of clustering quality. The root level is level zero and the first level is the children of the root. As CLIMIS follows the same clustering approach as Cobweb, and therefore also seeks clustering trees in which the first level is optimal, we only apply the *cutoff* operator to tree levels below the first level.

3.3 Contributions

This thesis set out to investigate if it is possible to scale conceptual clustering to large categorical data sets (see chapter 1). The contributions of this thesis with regard to scalability are as follows:

- Proposes CLIMIS, a scalable clustering algorithm for categorical data. CLIMIS is the only conceptual clustering algorithm that uses a compact cluster representation, its cluster representation requires minimum memory space and updating to incorporate new data, while maintaining the quality of the clustering.

- Shows an empirical evaluation of the scalability of CLIMIS by comparing it to i) Cobweb, and ii) categorical data algorithms developed for large data sets: a) LIMBO and b) ROCK. We demonstrate that CLIMIS shows better scalability compared to Cobweb, LIMBO and ROCK and, unlike ROCK, it

does not need sampling to scale.

- Shows an empirical evaluation of the quality of CLIMIS clustering by comparing it to the quality of (i) Cobweb, (ii) LIMBO, and (iii) ROCK.

Chapter 4

Evaluation of CLIMIS

This chapter presents an evaluation of CLIMIS using a comparative experimental analysis. CLIMIS is evaluated against other algorithms for its quality and ability to scale to large data sets.

4.1 Algorithms

We have used three algorithms in our evaluation. These algorithms are: (i) Cobweb (Fisher, 1987), (ii) LIMBO (Andritsos et al., 2004) and (iii) ROCK (Guha et al., 2000).

Cobweb

We have used Cobweb in our evaluation because Cobweb is the algorithm on which CLIMIS is based. In particular, we have used Weka's implementation of Cobweb (Garner, 1995; Witten & Frank, 2000, 2005). Weka is a widely used data mining environment that includes a number of different data mining algorithms.

One of these algorithms is Cobweb. Weka's Cobweb is an implementation of the algorithm in Fisher (1987). It implements the four operators - *incorporate*, *disjunct*, *split* and *merge* - to build and correct a clustering tree and evaluates these operators against each other with the use of the *category utility*. To support clustering of more data, Weka uses the *cutoff* parameter in the same way as CLIMIS (Witten & Frank, 2005).

LIMBO

We have used LIMBO in our evaluation because it is a recent algorithm developed for the analysis of large categorical data. The implementation of LIMBO was provided by the developer of the algorithm, Periklis Andritsos. The implementation includes all three phases of the algorithm (discussed in chapter 2, section 2.6.3): (i) the summarisation of the data, (ii) the AIB algorithm, and (iii) the linking of the records to the clusters. The algorithm makes use of one of two parameters, ϕ or S, (also discussed in chapter 2, section 2.6.3), to scale to large data. In our evaluation, we only made use of the ϕ parameter when we compared LIMBO's scalability to CLIMIS. We avoided the S parameter as it was not been fully implemented in the implementation provided.

LIMBO was considered a good algorithm to use in the evaluation of CLIMIS because it has similar characteristics with CLIMIS. These characteristics are the following:

1. LIMBO is an algorithm developed for categorical data that cannot be ordered,

2. it uses a measure that can discover the natural clusters in categorical data,

3. it is a hierarchical algorithm, and

4. it can be applied to large data sets of categorical data.

ROCK

We have used ROCK in our evaluation because it is a widely cited algorithm developed for the analysis of large categorical data. ROCK relies on sampling to scale to large data sets. We used *random sampling without replacement* to select a sample for ROCK (Han & Kamber, 2000). We used 1% of the original data set in the sample following the suggestion in Guha et al. (2000).

The implementation of ROCK used in this work has been provided by the developer Guha et al. (2000). The implementation includes only one phase of the algorithm, which is the phase that builds a clustering based on a sample (see chapter 2, section 2.6.2). The second phase of the algorithm, which assigns the rest of the data to the clusters produced from phase 1 is not included. We have implemented the second phase of the algorithm following the algorithm described by Guha et al. (2000) in order to test the overall performance of ROCK's clustering.

The current implementation of ROCK as presented in Guha et al. (2000) uses a metric measure and requires categorical data to be transformed to binary. This thesis used a simple conversion algorithm developed to fit the ROCK algorithm that converts categorical data to the required binary representation. The conversion process was not part of the evaluation.

ROCK was considered a good algorithm to use in the evaluation of CLIMIS because it has similar characteristics with CLIMIS. These characteristics are the

following:

1. ROCK is an algorithm developed for categorical data that cannot be ordered,
2. it uses a measure that can discover the natural clusters in categorical data,
3. it can be applied to large data sets of categorical data, and
4. it is a hierarchical algorithm.

4.2 Data Sets

We have used real and synthetic data sets in our evaluation. The real data sets are: (i) congressional votes, and (ii) mushroom. These data sets were taken from the UCI machine learning repository [1]. We selected these two real data sets because they are often used for categorical data analysis (Andritsos et al., 2004; Barbará et al., 2002; Biswas et al., 1998; Ganti et al., 1999; Guha et al., 2000; Zaki et al., 2005).

Congressional Votes

The congressional votes data set contains U.S. House of Representatives votes on 16 issues from 1984:

1. handicapped-infants
2. water-project-cost-sharing

[1] http://mlearn.ics.uci.edu/databases/

3. adoption-of-the-budget-resolution
4. physician-fee-freeze
5. el-salvador-aid
6. religious-groups-in-schools
7. anti-satellite-test-ban
8. aid-to-nicaraguan-contras
9. mx-missile
10. immigration
11. synfuels-corporation-cutback
12. education-spending
13. superfund-right-to-sue
14. crime
15. duty-free-exports
16. export-administration-act-south-africa

Every tuple in the data set represents the votes of a congressperson in the form of yes/no values. It contains 435 tuples each one classified as republican or democrat. There are 168 republicans and 267 democrats. The data set has a number of missing values in most of the attributes (see table 4.1). The missing

Attribute	Missing Values
1	12
2	48
3	11
4	11
5	15
6	11
7	14
8	15
9	22
10	7
11	21
12	31
13	25
14	17
15	28
16	104

Table 4.1: Missing Values

values were treated as another yes/no value. We checked our approach by comparing the clustering result we produced using ROCK with the result in Guha et al. (2000) and it was very similar.

Mushroom

The mushroom data set contains data about gilled mushrooms in the Agaricus Lepiota Family. Each entry is described according to 22 attributes that describe the colour, shape and habitat and class of the mushrooms. The attributes all contain nominal values (see table 4.2). There are 8124 tuples in the data set each one classified as edible or poisonous. There are 4208 edible and 3916 poisonous mushrooms. The data set has 2480 missing values denoted by *?* and they are all in the attribute 11. We treated the missing values as another value.

Attribute	Values
class	edible=e, poisonous=p
cap-shape	bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
cap-surface	fibrous=f, grooves=g, scaly=y, smooth=s
cap-color	brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p,purple=u,red=e, white=w, yellow=y
bruises?	bruises=t, no=f
odor	almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m,none=n,pungent=p,spicy=s
gill-attachment	attached=a, descending=d, free=f, notched=n
gill-spacing	close=c, crowded=w, distant=d
gill-size	broad=b, narrow=n
gill-color	black=k, brown=n, buff=b, chocolate=h, gray=g, green=r, orange=o, pink=p, purple=u, red=e, white=w, yellow=y
stalk-shape	enlarging=e, tapering=t
stalk-root	bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
stalk-surface-above-ring	ibrous=f, scaly=y, silky=k, smooth=s
stalk-surface-below-ring	ibrous=f, scaly=y, silky=k, smooth=s
stalk-color-above-ring	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p,red=e,white=w,yellow=y
stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p, red=e, white=w, yellow=y
veil-type	partial=p, universal=u
veil-color	brown=n, orange=o, white=w, yellow=y
ring-number	none=n, one=o, two=t
ring-type	cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
spore-print-color	black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
population	abundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
habitat	grasses=g, leaves=l, meadows=m, paths=p, urban=u,waste=w,woods=d

Table 4.2: Description of the Mushroom Data Set

Synthetic Data Sets

To evaluate the scalability of CLIMIS, we also used a synthetic data set generator called datgen [1]. This data set generator is the same generator that Andritsos (2004) and Barbará et al. (2002) have employed in their work. The generator produces data sets with varying number of tuples, attributes, attribute domains and classes (clusters). To produce a data set, the user simply specifies the desired number of tuples, attributes, classes (clusters) and the attribute domain size. The number of classes (clusters) in a data set can be specified using conjunctive rules of the type: $att1 = a1 \wedge att2 = a2 \wedge \ldots \Rightarrow class = c1$.

The data sets produced for this work were created considering the fact that the algorithms used in the experiments had different ability to scale. Two sets of groups of data sets were created. The first set was created to be used with algorithms that cannot scale:

- Group 1: the data sets contain 10 attributes, 10 clusters and varying number of tuples: 500 to 2500 tuples. The number of tuples was increased by 500 tuples.
- Group 2: the data sets contain 1000 tuples, 10 clusters and varying attributes: 10 to 50 attributes. The number of attributes was increased by 10 attributes.
- Group 3: the data sets contain 10 attributes, 10 clusters and varying number of tuples: 5,000 to 25,000 tuples. The number of tuples was increased by 5,000 tuples.

[1] See appendix D for a simple example of a data set produced with datgen.

The second set was created to be used in large scale experiments:

- Group 4: the data sets contain 10 attributes, 10 clusters and varying number of tuples: 25,000 to 150,000 tuples. The number of tuples was increased by 25,000 tuples.

- Group 5: the data sets contain 100,000 tuples, 10 clusters and varying number of attributes: 10 to 50 attributes. The number of attributes was increased by 10 attributes.

- Group 6: the data sets contain 100,000 tuples, 10 attributes and varying number of classes (clusters): 10 to 50 classes (clusters). The number of classes (clusters) was increased by 10 clusters.

- Group 7: the data sets contain 10 attributes, 10 clusters and varying number of tuples: 50,000 to 800,000 tuples. From 200,000 tuples, the number of tuples was increased by 200,000.

The size of the attribute domains in all the data sets, described above, varied between 7 and 30. This variation as well as the attribute number variation is similar to the domain size and attribute number variation used in Andritsos (2004). With regard to tuple number variation, Andritsos (2004) used larger data sets to test the scalability of the LIMBO algorithm. LIMBO is a multi-phase algorithm and the data sets with varying number of tuples were used to test the performance of phase 1 of the algorithm only [1], whereas we evaluated LIMBO as a multi-phase algorithm as phase 1 on its own does not produce a usable set

[1] See section 2.6.3 for a discussion of the different phases of LIMBO. Also, figure 4.5, page 89, shows the performance of LIMBO phase 1 and LIMBO as a multi-phase algorithm.

of clusters. The variation in number of clusters we used here is the same as the variation used by Andritsos (2004). Barbará et al. (2002) shows information about the attribute and tuple number variation used, which is similar to that of Andritsos (2004).

4.3 Quality

We evaluated the quality of the clusters produced by CLIMIS by following the evaluation approach used in Guha et al. (2000). In their paper, they evaluate the quality of clustering produced by ROCK using a well known data set with clear clusters. They applied the ROCK algorithm on the data set to see if ROCK could identify the known clusters. The clearer the clusters ROCK produced, and closer to the known clusters, the better the quality of the algorithm.

In our experiment, we also employed well known data sets with clear clusters. To evaluate the quality of the CLIMIS algorithm, we applied CLIMIS and three other algorithms - Cobweb, LIMBO and ROCK - on the chosen data sets. Two things were of interest in this experiment to evaluate quality:

1. how well CLIMIS identified the known clusters, and
2. how it compared to the other algorithms in that respect.

The data set used in the evaluation of Guha et al. (2000) is the congressional votes data set. An analysis of the frequency of the yes/no values against the two classes (democrat and republican) showed that there are two clear clusters in the congressional votes data set (see table 4.3). As table 4.3 shows, on 12 issues the majority of democrats voted differently to republicans. On only 3 issues

Cluster 1 (Republicans)	Cluster 2 (Democrats)
(immigration,y,0.51) (export-administration-act-south-africa,y,0.55) (synfuels-corporation-cutback,n,0.77)	(immigration,y,0.51) (export-administration-act-south-africa,y,0.7) (synfuels-corporation-cutback,n,0.56)
(adoption-of-the-budget-resolution,n,0.87) (physician-fee-freeze,y,0.92) (el-salvador-aid,y,0.99) (religious-groups-in-schools,y,0.93) (anti-satellite-test-ban,n,0.84) (aid-to-nicaraguan-contras,n,0.9) (mx-missile,n,0.93) (education-spending,y,0.86) (crime,y,0.98) (duty-free-exports,n,0.89) (handicapped-infants,n,0.85) (superfund-right-to-sue,y,0.9) (water-project-cost-sharing,y,0.51)	(adoption-of-the-budget-resolution,y,0.94) (physician-fee-freeze,n,0.96) (el-salvador-aid,n,0.92) (religious-groups-in-schools,n,0.67) (anti-satellite-test-ban,y,0.89) (aid-to-nicaraguan-contras,y,0.97) (mx-missile,y,0.86) (education-spending,n,0.9) (crime,n,0.73) (duty-free-exports,y,0.68) (handicapped-infants,y,0.65) (superfund-right-to-sue,n,0.79)

Table 4.3: Congressional Votes Frequency Analysis (Guha et al., 2000)

the majority vote is the same for both democrats and republicans. Within both groups, democrats and republicans, the majority vote (yes or no), on each of the 12 issues, is quite high. This is evidence that there are two clear clusters within the data set that have high intra-cluster similarity and inter-cluster dissimilarity. This is also supported by the evaluation by Andritsos (2004) on the congressional votes data set. Andritsos used various measures to identify the most natural clustering size for a given data set and his conclusion was that there are two natural clusters in congressional votes. We used the congressional votes data set to compare CLIMIS to Cobweb (the Cobweb implementation in the Weka environment), ROCK and LIMBO.

The number of clusters produced by LIMBO and ROCK is determined by a user specified parameter and not by the algorithm. Cobweb and CLIMIS, on the other hand, return the best clustering the algorithm can produce from the data set. Furthermore, LIMBO and ROCK return a flat organisation of clusters while

Table 4.4: Congressional Votes - 2 Clusters

COBWEB - Level 2 Clusters		
Cluster No.	No. of Democrats	No. of Republicans
1	45	158
2	222	10
ROCK - θ = 0.7		
Cluster No.	No. of Democrats	No. of Republicans
1	12	132
2	255	36
LIMBO		
Cluster No.	No. of Democrats	No. of Republicans
1	53	161
2	214	7
CLIMIS - Level 2 Clusters		
Cluster No.	No. of Democrats	No. of Republicans
1	44	159
2	223	9

Cobweb and CLIMIS return a hierarchical organisation of clusters. To compare the results of Cobweb and CLIMIS with the other algorithms, we used the clusters from level 2 in the Cobweb output as that is considered by Fisher (1987) to be the best level.

Table 4.4 shows the clustering results produced after applying Cobweb, ROCK, LIMBO and CLIMIS on congressional votes. There are 168 republicans and 267 democrats in congressional votes and as the table shows all four algorithms have produced clear clusters, one containing a majority of republicans and the other containing a majority of democrats. ROCK has produced a republicans cluster (cluster 1) with only 12 democrats. LIMBO, CLIMIS and Cobweb have produced between 44 and 53 democrats in that cluster. ROCK has produced the worst democrats cluster (cluster 2) with 36 republicans. The democrats cluster produced by the other algorithms is similar as it has 7 to 10 republicans.

We can see that Cobweb and CLIMIS produced almost identical results, which was expected. Both algorithms were run using zero *cutoff*, they implement the same operators and use the category utility measure in their evaluation approach. The small difference in the results is explained by the fact that they are different implementations. For example, Fisher's description of the algorithm does not state how the *best operator* is selected when more than one operator produces the same category utility. This is left to the implementation. An interesting observation is that both algorithms discovered two clusters at level 2, matching the natural clustering of the data.

The ROCK algorithm has been developed to rely on sampling but as the data set used in this experiment is small, we used the entire data set. ROCK had to be applied a number of times in order to produce a good clustering. ROCK's quality depends upon the θ threshold (see section 2.6.2). Two records can only be considered neighbours if their similarity is above the θ threshold. We noticed that the algorithm is quite sensitive to the value of θ. A large or small θ can lead to clusters of poor quality. Appendix A shows the results of applying ROCK on the same data set with varying θ. As it is clear from the appendix and is also discussed in Barbará et al. (2002), ROCK requires considerable tuning in order to produce a good clustering.

The LIMBO algorithm is a multi-phase algorithm. The algorithm summarises the data in phase 1 using the ϕ parameter in order to achieve scalability in phase 2. When the ϕ parameter is increased, the algorithm tends to trade off quality for scalability. Phase 2 involves the application of the AIB algorithm. As congressional votes is a small data set, we set the ϕ parameter, which can range from 0 to 1, to 0. With $\phi = 0$ LIMBO applies the traditional AIB algorithm.

Table 4.5: Mushroom - 3 Clusters

ROCK - θ = 0.9		
Cluster No.	No. of Edible Mushrooms	No. of Poisonous Mushrooms
1	4208	2152
2	0	36
3	0	1728
LIMBO		
Cluster No.	No. of Edible Mushrooms	No. of Poisonous Mushrooms
1	4208	892
2	0	1296
3	0	1728
CLIMIS - Level 2 Clusters		
Cluster No.	No. of Edible Mushrooms	No. of Poisonous Mushrooms
1	4208	915
2	0	1312
3	0	1689

Therefore, the results we present in table 4.4 represent the best clustering quality the LIMBO algorithm can produce on congressional votes.

Apart from the congressional votes data set, we also used the mushroom data set to compare the clustering quality between CLIMIS, LIMBO and ROCK. Unfortunately, the size of the mushroom data set did not allow this data set to be used in the clustering quality comparison between CLIMIS and Cobweb as Cobweb cannot handle large data sets. Also, we had to tune the ROCK algorithm to find the θ value that produces the best clustering, which in this case was 0.9. The congressional votes experiment produced the best clustering with $\theta = 0.7$. Appendix B shows the results of applying ROCK on the mushroom data set with varying θ.

To draw our conclusions based on the mushroom data set, we relied on Andritsos (2004) analysis of the mushroom data set, which indicated that the most natural clustering size of the mushroom data is 3 clusters. Interestingly, when

we applied CLIMIS on the mushroom data set the algorithm produced 3 clusters at level 2. Table 4.5 shows the results of applying LIMBO, CLIMIS and ROCK on the mushroom data set. It is evident that CLIMIS and LIMBO produced the best edible mushrooms cluster (cluster 1) with 915 and 892 poisonous mushrooms in cluster 1 respectively. ROCK has almost twice as many poisonous mushrooms in the edible mushrooms cluster, 2152 poisonous mushrooms.

Using the Cutoff

The *cutoff* parameter provides a threshold, which the algorithm uses to decide whether a cluster is of any value. If the algorithm decides that the cluster below a current cluster C does not improve *category utility*, it removes the cluster from the tree. The *cutoff* parameter should have little impact on the quality of CLIMIS as the algorithm performs conceptual clustering, which favours level 2 and we only apply the parameter below level 2.

To see the impact that the *cutoff* parameter has on a clustering produced by CLIMIS, we applied CLIMIS on the congressional votes and mushroom data sets with and without the *cutoff*. As tables 4.6 and 4.7 show, the *cutoff* threshold has little impact on the quality of the clusters. The results shown in table 4.6 were obtained when we applied CLIMIS with *cutoff* $0.05 - 0.5$. The mushroom result is based on *cutoff* $= 0.05$.

4.4 Scalability

This section presents an evaluation of the scalability of CLIMIS by comparing it with Cobweb, LIMBO and ROCK. In all three cases, the scalability of the

Table 4.6: Congressional Votes - Using Cutoff

CLIMIS - Level 2 Clusters Using Cutoff		
Cluster No.	No. of Democrats	No. of Republicans
1	41	158
2	226	10
CLIMIS - Level 2 Clusters		
Cluster No.	No. of Democrats	No. of Republicans
1	44	159
2	223	9

Table 4.7: Mushroom - Using Cutoff

CLIMIS - Level 2 Clusters Using Cutoff		
Cluster No.	No. of Edible Mushrooms	No. of Poisonous Mushrooms
1	4208	915
2	0	1312
3	0	1689
CLIMIS - Level 2 Clusters		
Cluster No.	No. of Edible Mushrooms	No. of Poisonous Mushrooms
1	4208	915
2	0	1312
3	0	1689

algorithms has been compared with regard to (i) increasing the number of tuples, (ii) increasing the number of attributes, and (iii) increasing the number of clusters.

Our evaluation follows the approach used in the well cited scalable algorithm BIRCH (Zhang, 1997). Zhang (1997) applied BIRCH and other algorithms, with similar characteristics, on the same data sets that varied in: (i) number of tuples, (ii) number of attributes, and (iii) number of clusters. Zhang (1997) recorded the performance of the algorithms as the tuples in the data sets increased, then as the attributes in the data sets increased and finally as the clusters in the data sets increased.

In our evaluation of the CLIMIS scalability, we used the same variations

in the data sets to evaluate (i) how the performance of CLIMIS is affected from an increase in either tuples, attributes or clusters, and (ii) how the performance of CLIMIS compares to the other algorithms that have similar characteristics. CLIMIS was compared to Cobweb using one collection of data sets due to the limitation of Cobweb. With this comparison, we intended to show the difference that CLIMIS has made to the scalability of conceptual clustering. CLIMIS was compared to LIMBO and ROCK using another collection of data sets. With this experiment, we intended to show how well CLIMIS performs when compared to algorithms developed for large scale clustering of categorical data.

4.4.1 CLIMIS versus Cobweb

To compare CLIMIS to Cobweb, we used synthetic and real data. We performed three experiments. The first experiment used zero *cutoff* and tested the scalability of the algorithms with regard to the number of tuples (see figure 4.1). This experiment used relatively small data sets as with larger data sets Cobweb runs out of main memory. The real data sets were subsets of the mushroom data set and had smaller attribute domains than the synthetic data sets. All data sets used had 10 attributes. The second experiment, which also used zero *cutoff*, tested the scalability of the algorithms with regard to number of attributes (see figure 4.2). This experiment used synthetic and real data sets. The real data sets were created based on mushroom and had smaller attribute domains than the synthetic data sets. All the data sets had 1000 tuples. The third experiment (see figure 4.3) used $cutoff = 0.4$ and tested the scalability of the algorithms with regard to larger number of tuples. All the data sets in this experiment were

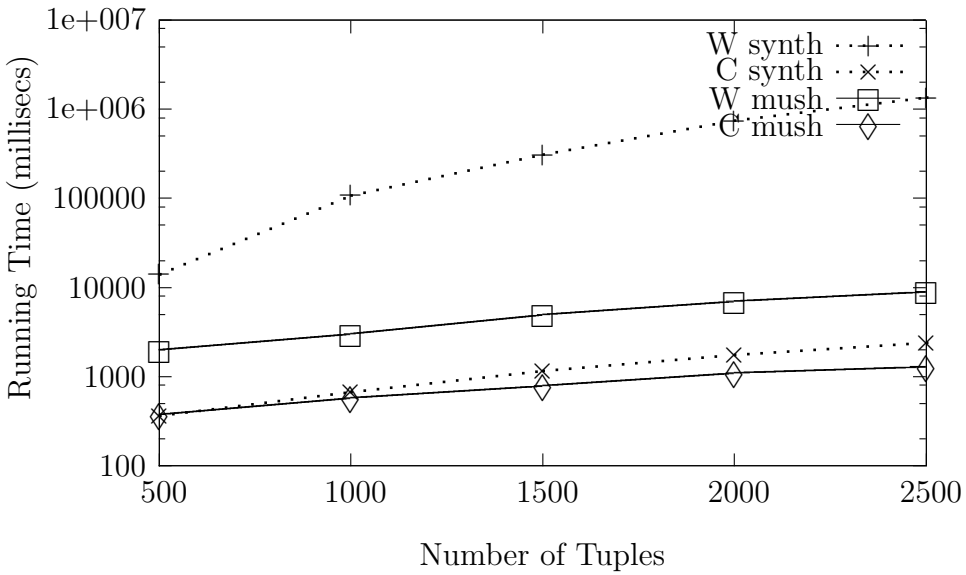

Figure 4.1: CLIMIS versus Weka (Cobweb): Increasing the number of tuples

synthetic and had 10 attributes.

As figures 4.1, 4.2 and 4.3 show, in all cases, CLIMIS outperformed Cobweb. In figures 4.1 and 4.2, both algorithms show better scalability on the mushroom data sets. The scalability on larger data sets using the *cutoff* threshold, shown in figure 4.3, is very different between the two algorithms. CLIMIS performance appears to have little fluctuation, whereas Cobweb becomes significantly slower as the size of data increases.

The reason CLIMIS shows better scalability than Cobweb is because it performs less cluster scans to calculate *category utility*. For example, when Cobweb computes *category utility* for the *incorporate* operator, it has to compute the sum of the squared conditional probabilities for each cluster at a level every time it tries a new tuple in each cluster. This involves a lot of main memory scans, which increase as more data is clustered. The size of the clusters and the number of

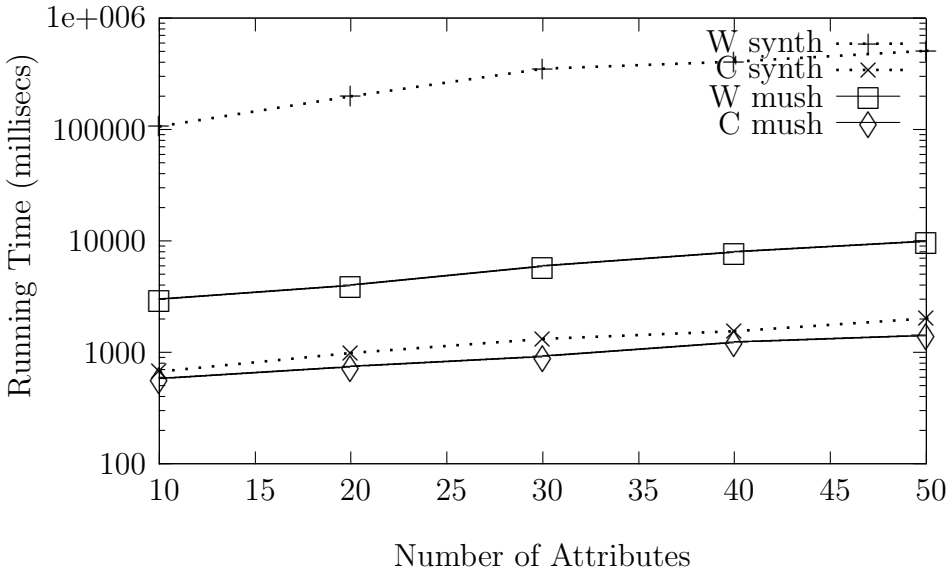

Figure 4.2: CLIMIS versus Weka (Cobweb): Increasing the number of attributes

clusters get larger as more data is clustered. An increase in the size of clusters means that Cobweb has to scan more conditional probabilities in a cluster. A larger number of clusters results in more attribute value probability reads.

The increase in the amount of data has smaller impact on CLIMIS's performance as CLIMIS uses a different cluster representation, which provides the sum of the squared conditional probabilities for every cluster.

4.4.2 CLIMIS versus LIMBO

The data sets we used to compare CLIMIS scalability to LIMBO were all synthetic. With regard to increasing number of tuples, we used data sets of a varying number of tuples with 10 clusters and 10 attributes. With regard to increasing number of attributes, we used data sets of a varying number of attributes with $100K$ tuples and 10 clusters. With regard to increasing number of clusters, we

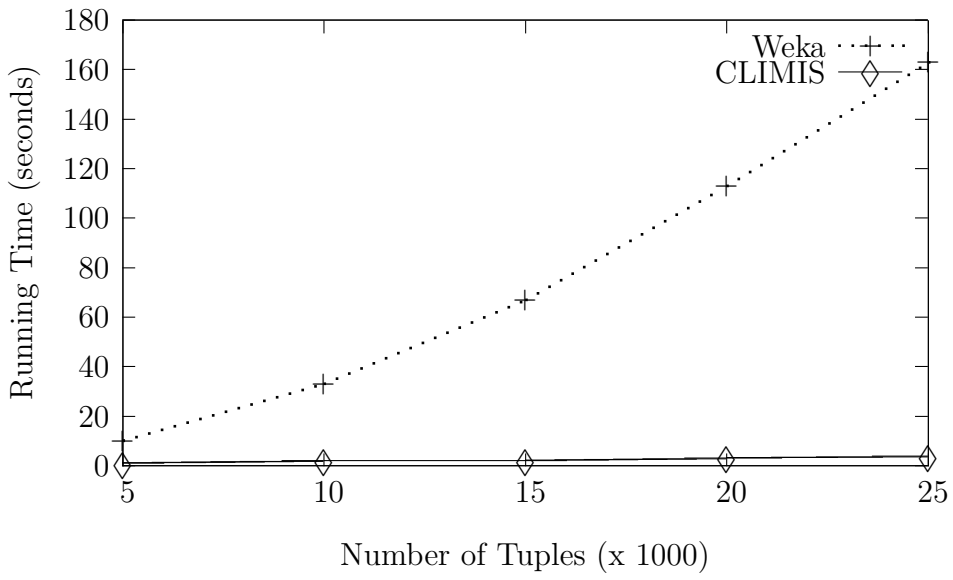

Figure 4.3: CLIMIS versus Weka (Cobweb): Using the *cutoff* parameter

used data sets of a varying number of clusters with $100K$ tuples and 10 attributes.

We used $cutoff = 0.4$ for CLIMIS as it produced reasonable clusters and varying ϕ and B parameters for LIMBO depending on the data set requirements. With regard to increasing the number of tuples, we ran a number of experiments with LIMBO in order to decide on the ϕ and B parameters as we aimed to use the same values for all trials to make the results more comparable. The value of ϕ was varied from 0 to 1.5 and the value of B from 4 to 50 [1]. We used $\phi = 0.7$ and $B = 4$. A larger B value produced only 1 cluster with data sets less than $100K$. As figure 4.4 illustrates, CLIMIS shows better performance than LIMBO. The LIMBO approach separates the summarisation from the clustering process. We tried to increase the number of tuples to more than $150K$ tuples but LIMBO failed to return any number of clusters as it either crashed or stopped. This was

[1] The same variations of the ϕ and B parameters were used by Andritsos (2004)

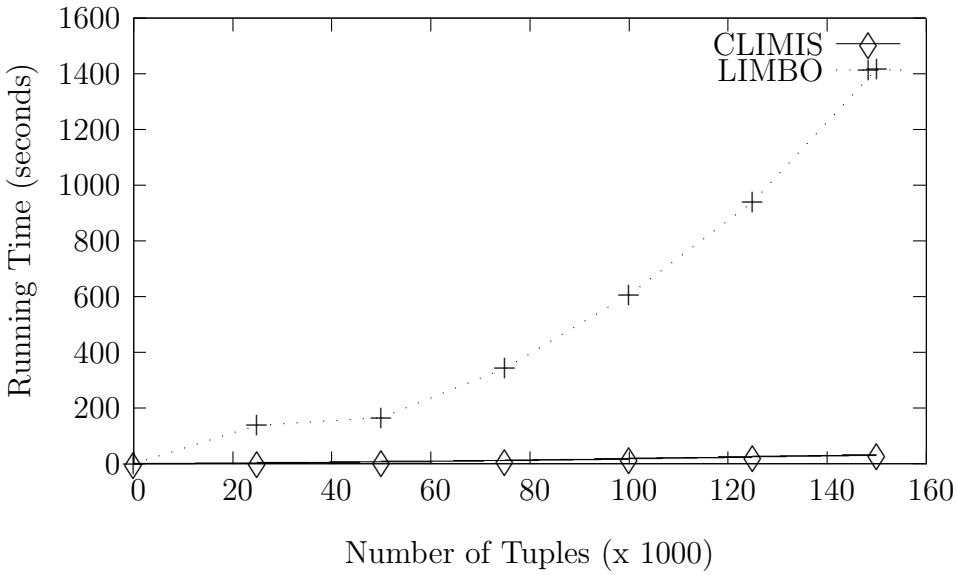

Figure 4.4: CLIMIS versus LIMBO: Increasing the number of tuples

despite trying different values of ϕ and B. With any more than $150K$ tuples, the ϕ value caused either over summarisation and the clustering result contained one cluster or not enough summarisation to complete the process within the available memory. Even with $100K$, it was difficult to tune the algorithm to return the specified clusters. Appendix C shows an example of the trials performed on $100K$.

Unlike LIMBO, CLIMIS summarises the data while clustering the data. The effect CLIMIS's approach has on its performance is shown in figure 4.4. It is evident that compared to LIMBO, CLIMIS has better performance. The CLIMIS approach supports better performance because it processes every record only once. LIMBO, on the other hand, has to process every record three times as it involves 3 phases (see chapter 2, section 2.6.3). Phase 3 only takes a few seconds in all cases. Phase 2, on the other hand, which is the application of the AIB algorithm, suffers from the complexity of the AIB algorithm and unless

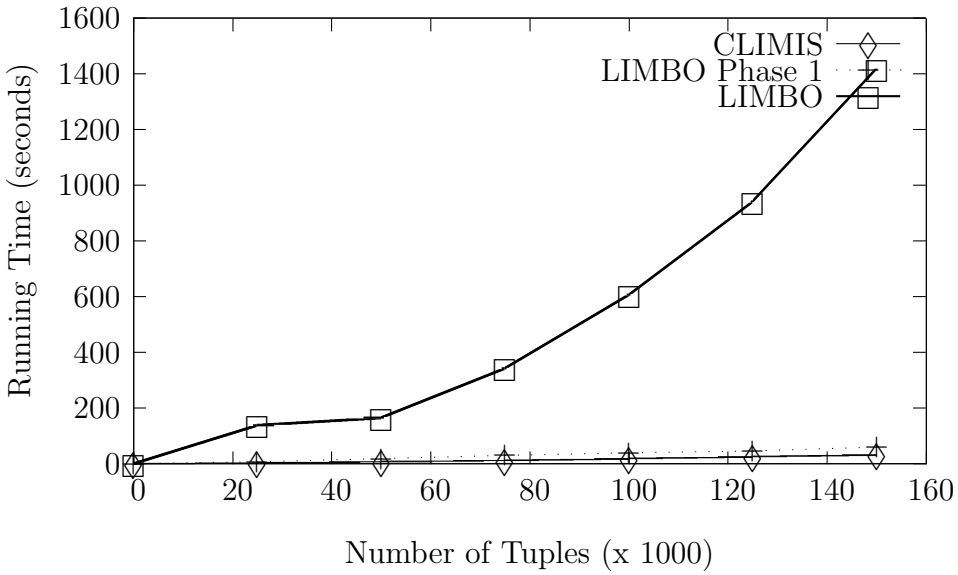

Figure 4.5: CLIMIS versus LIMBO Phase 1: Increasing the number of tuples

phase 1 is very successful in producing an appropriate number of nodes - not too many to allow the algorithm to scale and not too few to allow the algorithm to produce a 'good' clustering - the algorithm either fails to produce a clustering or produces a single cluster. The scalability of phase 1 of LIMBO is comparable to CLIMIS but phase 1 does not produce a set of usable clusters (see figure 4.5).

Figure 4.6 presents a comparison between LIMBO and CLIMIS with regard to increasing number of attributes. CLIMIS performance is better compared to LIMBO and can be predicted. LIMBO, on the other hand, is less predictable as less attributes may require more time to produce a clustering. For example, it consistently took the algorithm more time to produce a clustering with 10 attributes than with 20 attributes. The reason behind the performance of LIMBO is that we had to use different values of ϕ and B on different attribute sizes to achieve a clustering. The values of ϕ used were 0.7 to 0.9 and the values of B

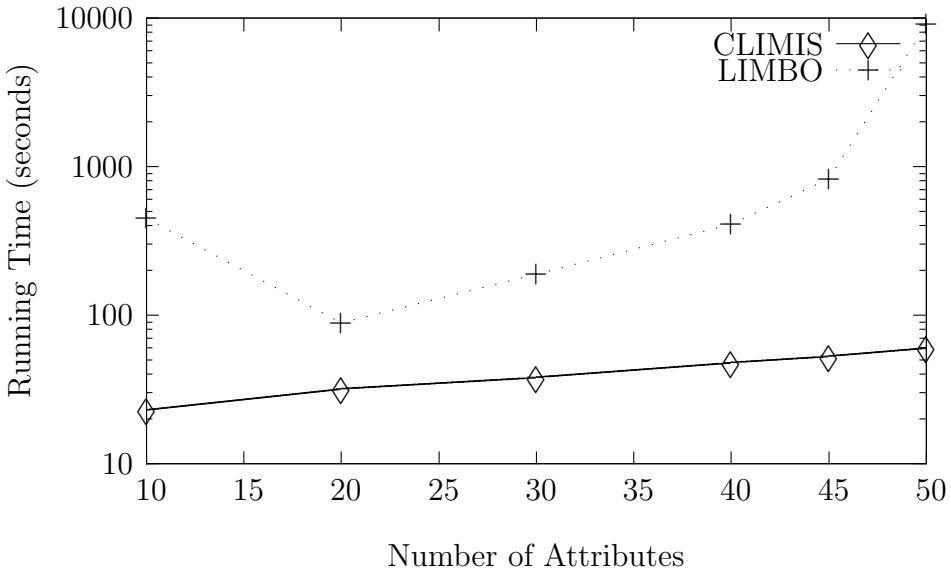

Figure 4.6: CLIMIS versus LIMBO: Increasing the number of attributes

varied from 4 to 50.

We applied CLIMIS and LIMBO to data sets with increasing clusters. The effect the increasing clusters had on the performance of CLIMIS was small. The time it took CLIMIS to complete the clustering process increased only a few seconds (see figure 4.7). The effect on the performance of LIMBO, however, was quite important. LIMBO failed to return a clustering when we run it on a data set with more than 10 clusters. This was despite running the experiment with varying values of ϕ and B. We have checked Andritsos (2004) for his results but, unfortunately, it includes no experiments that test the effect of increasing clusters on the algorithm's performance. We conclude that LIMBO is very sensitive to data sets with large number of clusters. A data set with a large number of clusters will require more splits in phase 1. More splits increase the number of nodes phase 1 produces making the AIB algorithm more expensive, which causes the failure

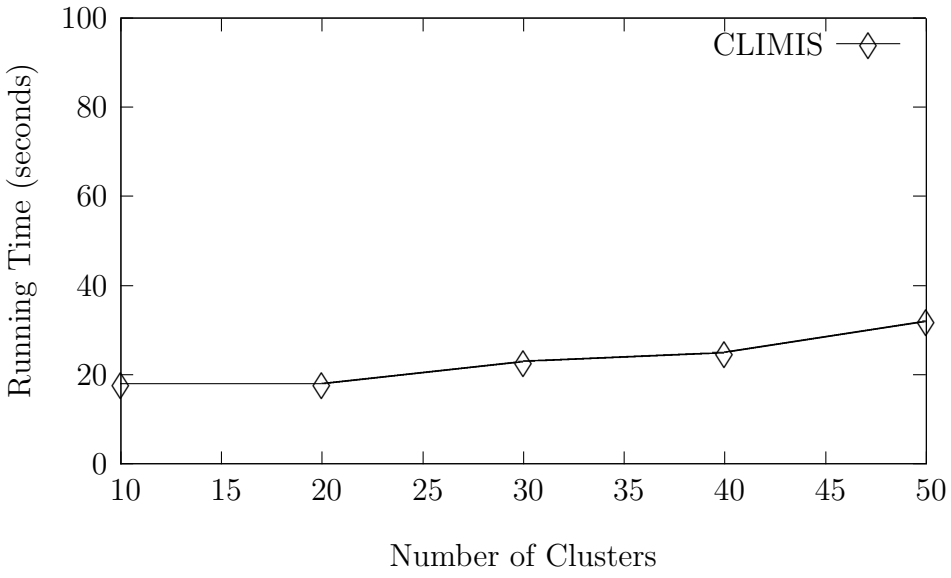

Figure 4.7: CLIMIS: Increasing clusters in the data set

of LIMBO to produce a clustering within the available memory.

4.4.3 CLIMIS versus ROCK

The data sets we used to compare the scalability of CLIMIS to ROCK were the same as the data sets used in the LIMBO comparison. With regard to increasing number of tuples, we used data sets of a varying number of tuples with 10 clusters and 10 attributes. With regard to increasing number of attributes, we used data sets of a varying number of attributes with $100K$ tuples and 10 clusters. With regard to increasing number of clusters, we used data sets of a varying number of clusters with $100K$ tuples and 10 attributes. We used $cutoff = 0.4$ for CLIMIS and varied θ $(0 - 1)$ for ROCK depending on the data set requirements. CLIMIS was applied on full data sets. ROCK in phase 1 was applied on samples drawn from the data sets and in Phase 2 the rest of the data sets were used.

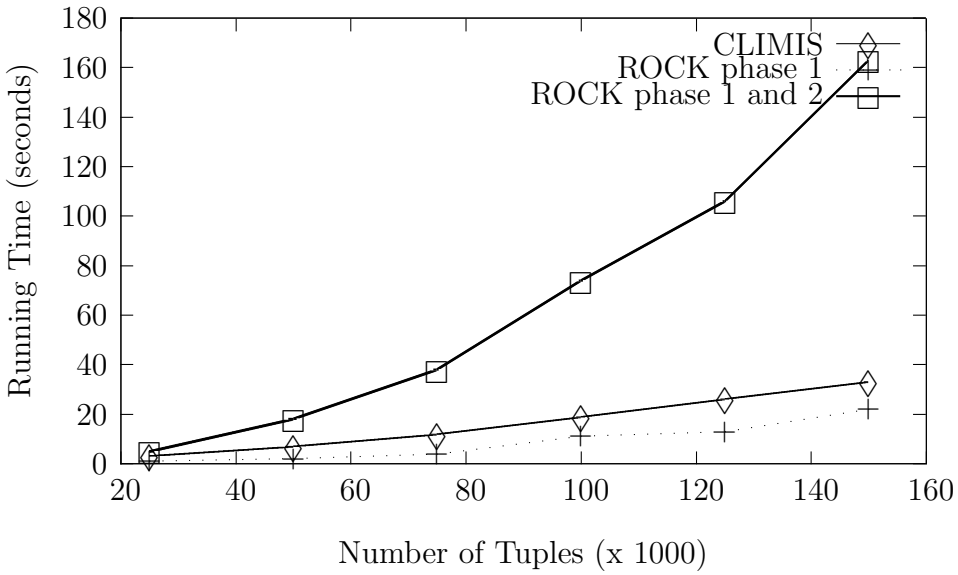

Figure 4.8: CLIMIS versus ROCK: Increasing the number of tuples

A comparison between CLIMIS and ROCK with regard to increasing number of tuples is illustrated in figure 4.8. As expected, phase 1 of ROCK is faster than CLIMIS. The experiment with phase 1 of ROCK involved quite small data sets as they were samples of the original data sets. The samples used were 1% of the original data sets so the largest sample with regard to number of tuples was about 1500 tuples. Figure 4.8 also shows a comparison between CLIMIS and the overall performance of ROCK. It is evident that CLIMIS shows better performance than ROCK. Also, phase 2 adds a large overhead to ROCK's performance.

ROCK phase 1 involves more complex operations compared to ROCK phase 2. Phase 1 has to evaluate the similarity of all pairs of points before starting to merge the clusters while phase 2 compares a point to a few extracted points from the clusters (we used 10% of the points) to decide the best cluster the point

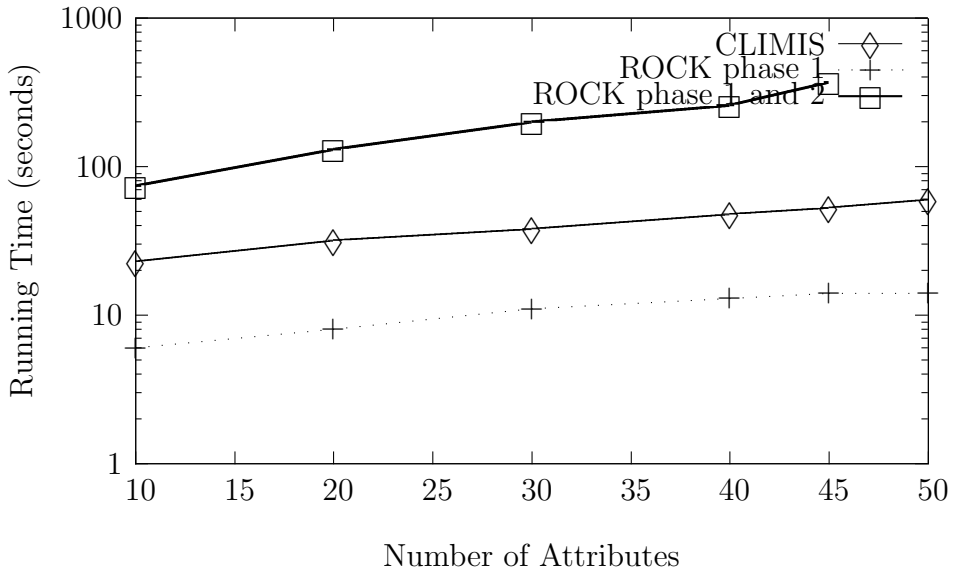

Figure 4.9: CLIMIS versus ROCK: Increasing the number of attributes

should be assigned. The only reason that phase 1 takes so little time compared to the overall performance of ROCK is that the data set used in phase 1 is of a small size.

With regard to increasing the number of attributes, as figure 4.9 illustrates, the performance of ROCK is better in phase 1 and worst overall compared to CLIMIS. However, an interesting observation is that we did not manage to complete the experiment with ROCK. When the algorithm was applied to 50 attributes it run out of main memory. The current implementation of ROCK involves a metric measure so we had to convert the categorical data to binary. The conversion created a very large number of attributes. The increase in number of attributes increases the number of operations that the algorithm has to perform. To compute the similarity between two points, the algorithm has to compute the distance between the points according to their dimensions. The larger the num-

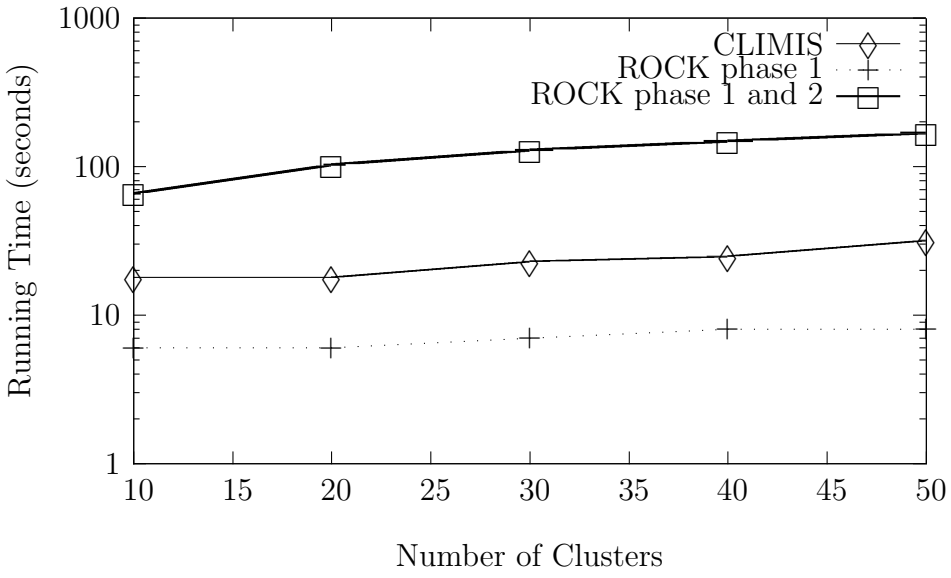

Figure 4.10: CLIMIS versus ROCK: Increasing the number of clusters

ber of dimensions, the more comparisons the algorithm has to perform, which explains the result shown in figure 4.9.

The effect the increase in number of clusters had on ROCK's performance was less compared to the increase in number of attributes (see figure 4.10). The increase in number of clusters caused more distinct attribute values in the data set. However, there were less attribute values and the conversion to binary created a smaller data set. The largest binary data set used in figure 4.10 had 252 attributes, whereas the largest binary set used in figure 4.9 had 500 attributes. Therefore, the algorithm had to perform less operations.

When comparing ROCK's overall performance to CLIMIS in figure 4.10, CLIMIS has better performance despite the fact that it does not use sampling.

With CLIMIS being a simpler and more scalable algorithm, we also tried it on larger data sets with regard to increasing the number of tuples (see figure

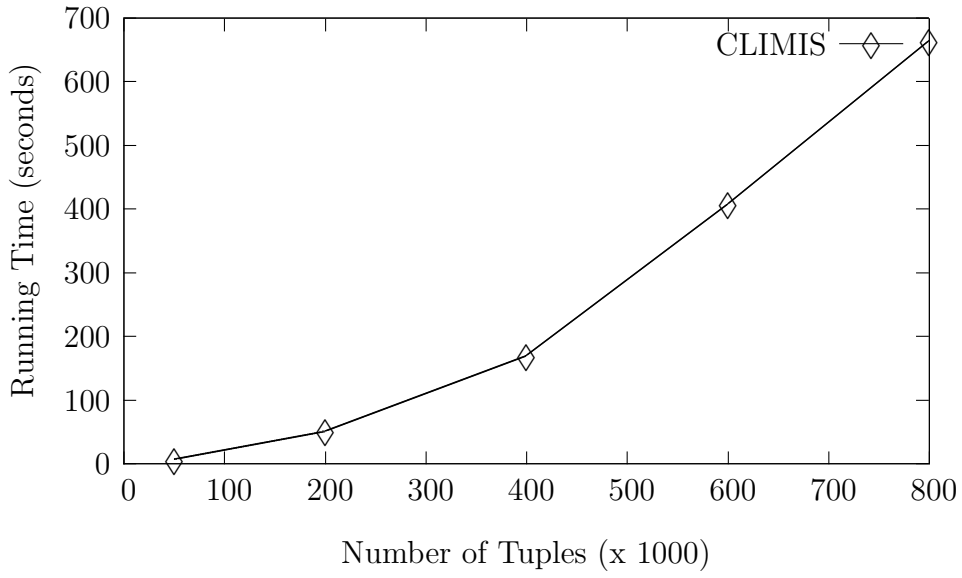

Figure 4.11: CLIMIS: Increasing the number of tuples

4.11). With larger data sets the algorithm appears to show the same performance that LIMBO and ROCK have on smaller data sets.

4.5 Discussion

The complexity of LIMBO relates to the way the algorithm is applied. LIMBO requires tuning to find a good combination for the ϕ and B parameters when it is applied to large data sets. The algorithm completes the first phase successfully in a reasonable time but it is not always successful with phase 2. Unfortunately, it may take LIMBO considerable time before it stops without returning a clustering result because of a main memory limitation. Appendix C shows the results of running LIMBO on a data set of $100K$ tuples with variations of ϕ. The algorithm produced a clustering only when $\phi = 0.7$. In all the other cases, the algorithm

did not produce a clustering and in almost half of those cases it took considerable time before it stopped. Because of the LIMBO approach, it took considerably more time to find a good parameter setting than the time it took the algorithm to complete a successful trial - it took hours to find a good parameter setting, whereas most of the trials took minutes to complete. Also, in our experiments, we knew the clusters in the data sets. Applying the algorithm to data sets that do not have known clusters would be difficult as it is not obvious when the optimal parameter values have been set. An advantage of the LIMBO algorithm is that it scaled the AIB algorithm to large categorical data sets.

The complexity of the ROCK algorithm also relates to the way it is used. There are two aspects of the algorithm that make its application complex: (i) the algorithm's parameters, and (ii) the algorithm's multi-phase nature. The current implementation of ROCK also adds to the application complexity as the data may have to be converted to binary.

ROCK requires two user defined parameters: the θ threshold parameter and the number of desired clusters. The θ parameter determines when two instances are considered similar as their similarity has to be larger than θ. The tuning process to find the best θ is not as difficult in phase 1 when the number of clusters in the data set is known in advance. When the number of clusters in the data set is not known in advance, the process becomes quite complex mainly because phase 1 must produce a good set of clusters for the successful completion of phase 2. An advantage of the ROCK algorithm is that it uses sampling and it can return a first set of clusters from phase 1 in minimum time.

The experiments show that CLIMIS is more scalable and simpler in its application for the following reasons:-

- The algorithm involves no data conversion from one data type to another.
- It does not require as much knowledge of the most natural clustering size in a data set as it builds the best clustering it can produce based on *category utility*.
- It does not have multiple phases that might depend on each other to produce good results or better performance.

Chapter 5

Implementation

The current version of CLIMIS has been implemented using java. The correctness of the implementation was confirmed when the quality of the CLIMIS output was compared to that of Cobweb shown in section 4.3. Furthermore, the computed *category utility* as well as CLIMIS's selection of *best operator* were compared, at every step, with results reached by applying the algorithm on small data sets manually. At every step, CLIMIS's results matched the results reached manually.

This chapter presents the real world structure of CLIMIS in a number of packages and, class diagrams that reflect different aspects of the algorithm's implementation (section 5.1). The chapter also discusses the input that CLIMIS receives and the output that the algorithm produces (section 5.2).

5.1 Logical Architecture Packages

There are three logical packages in CLIMIS: (i) Data Clustering, (ii) Prediction, and (iii) Data Management.

5.1.1 Data Clustering

Data clustering is the core package of CLIMIS. It performs the clustering evaluation and builds the clustering tree. Data clustering contains the following parts:

1. Cluster: It provides operations for the calculation of *category utility* (CU) and the evaluation of the four operators: (i) *incorporate*, (ii) *disjunct*, (iii) *split*, and (iv) *merge*. It decides the *best CU* and implements an operator accordingly by calling for changes in the *Tree* and, if necessary, the *PD-structure* (see figure 5.1).

2. Tree: It manages any updates to the clustering tree, for example, the addition of a new node to the clustering tree due to the implementation of the *disjunct* operator. Figure 5.2 is a break down of the *Tree* design.

3. PD-structure: It provides the conditional probability distribution of a leaf cluster. It manages any updates to the *PD-structure*, for example, the removal of a leaf cluster that has ceased being a leaf cluster. Figure 5.3 is a break down of the *PD-structure* design.

5.1.2 Prediction

Prediction is another package of CLIMIS. This package allows the clustering of tuples with missing data using an existing clustering tree for prediction purposes. Prediction includes the following parts:

1. Predict: It provides operations for the calculation of *category utility* for the *incorporate* operator. It decides the *best CU* and incorporates a tuple to

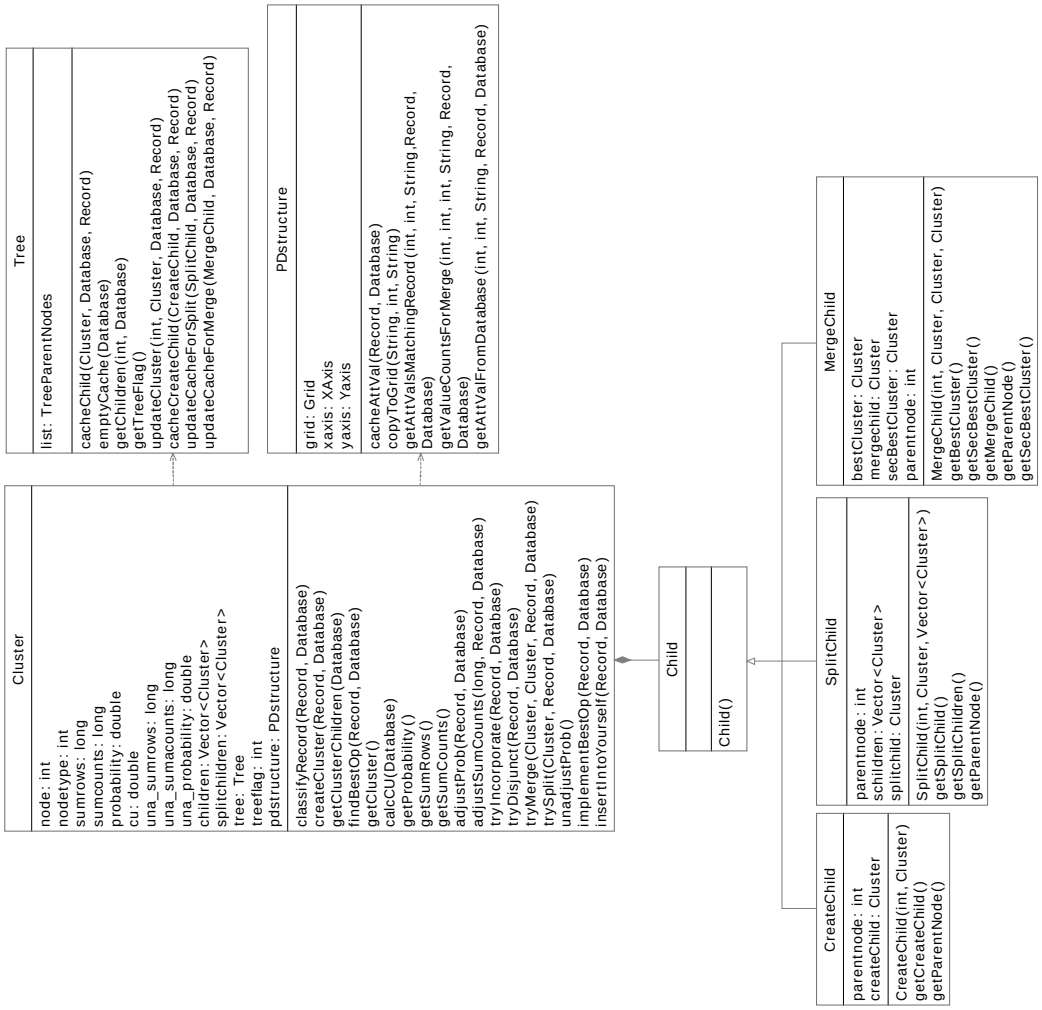

Figure 5.1: Data Clustering - Class Diagram

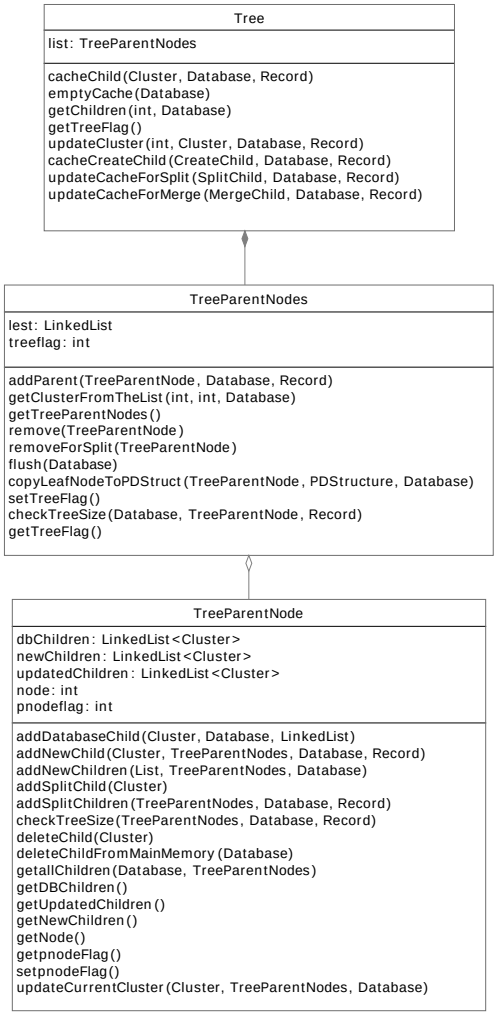

Figure 5.2: Tree - Class Diagram

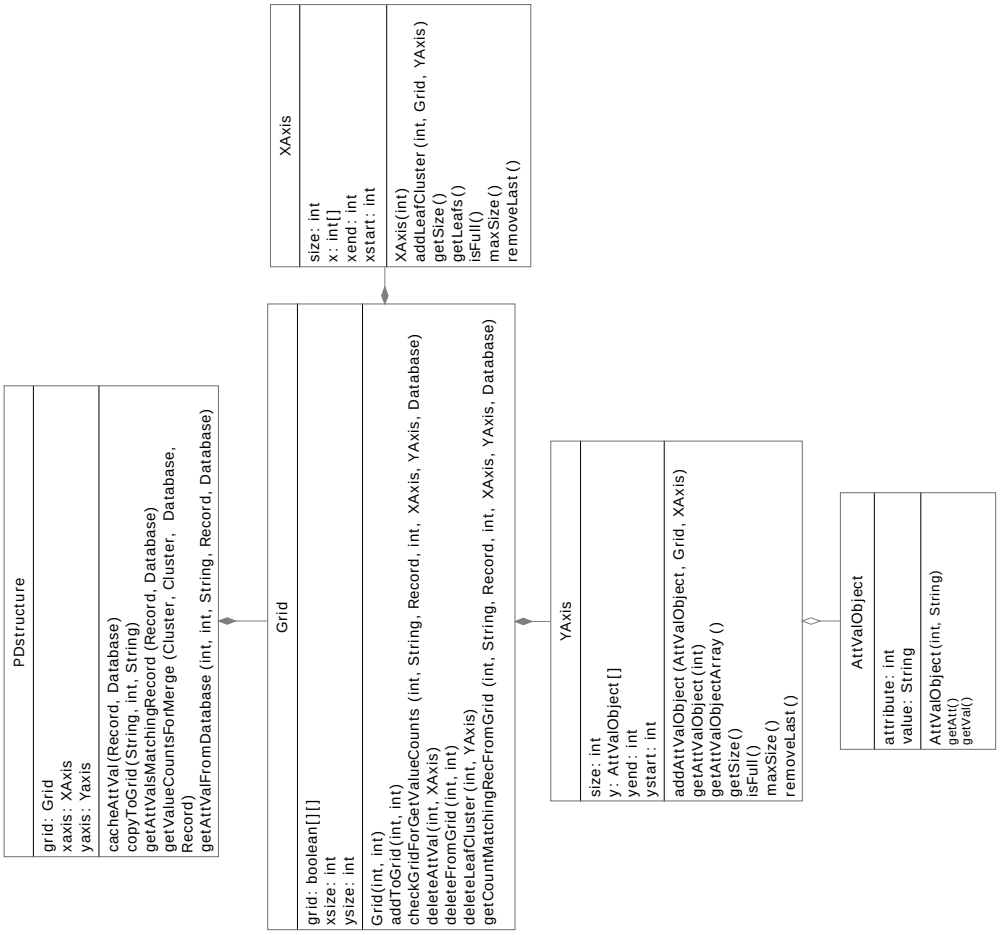

Figure 5.3: PD-structure - Class Diagram

the cluster indicated by the *CU*. *Predict* is the same as *Cluster* without the *disjunct*, *split* and *merge* operators and without changing the value of a cluster when *incorporate* happens.

5.1.3 Data management

The data management package manages any data that is read from or written to a DBMS and data that is read from a file. This package has the following parts:

1. Record: It provides a new tuple to be clustered.
2. Database: It is the interface between the data clustering as well as prediction package and the DBMS.
3. Parameter: It provides the parameters that the algorithm uses. For example, the *cutoff* parameter and the size of the *CLIMIS tree*.

Figure 5.4 shows a class diagram of the data management aspect of the CLIMIS implementation.

5.2 Input/Output

5.2.1 Input

One input to the CLIMIS algorithm is the raw data. This input comes from a database relation that may be stored in any relational DBMS. CLIMIS, also, takes one file as its input called *parameters* that contains the parameter settings for running the algorithm. The parameters that can be specified in the file include the following:

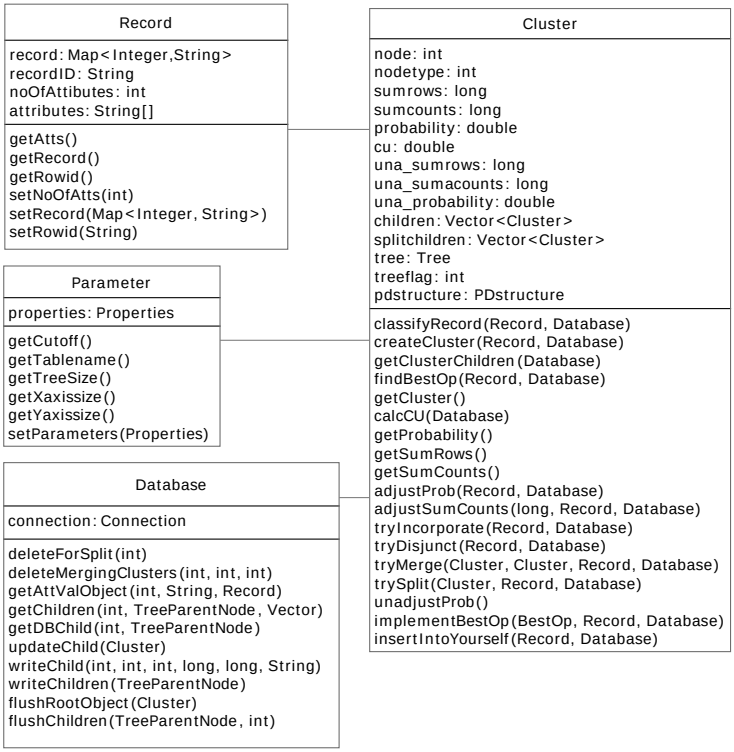

Figure 5.4: Data Management - Class Diagram

1. The name of the database relation that contains the tuples to be clustered.

2. The size of the *CLIMIS tree* structure.

3. The size of the *PD-structure*.

4. The size of the cutoff parameter.

5.2.2 Output

CLIMIS stores its output in database relations. There are two relations that store the output of the data clustering package: (i) the tree relation, and (ii) the cluster_content relation.

1. The tree relation, $tree(Parent, Child, Type, Probability, SUMSQ)$. This relation stores the clusters. A cluster is represented with its parent's identifier, its own identifier, its type (leaf cluster or internal cluster), its probability and its *cluster summarisation (SUMSQ)* (see figure 5.5).

2. The cluster content relation, $cluster\_content(Leaf, Recid)$. This is a relation that relates a leaf cluster to the original tuples (see figure 5.5). Only the leaf clusters are related to the tuples they cover as the other clusters can be related to tuples through the tree relation.

Tree Relation

Parent	Child	Type	Probability	SUMSQ
C_0	C_0	internal	$p(C_0)$	$SUMSQ_0$
C_0	C_1	internal	$p(C_1)$	$SUMSQ_1$
C_0	C_2	internal	$p(C_2)$	$SUMSQ_2$
C_1	C_3	leaf	$p(C_3)$	$SUMSQ_3$
C_1	C_4	leaf	$p(C_4)$	$SUMSQ_4$
C_2	C_5	leaf	$p(C_5)$	$SUMSQ_5$
C_2	C_6	leaf	$p(C_6)$	$SUMSQ_6$

Cluster_Content Relation

Leaf	Recid
C_3	r1
C_3	r2
C_4	r3
C_5	r4
C_6	r5
C_6	r6
C_6	r7

Figure 5.5: Relations Used by CLIMIS

Chapter 6

Integrating Data Mining with the DBMS

Data mining is used in areas where large volumes of data are collected, such as marketing, telecommunications, biology and the world wide web. Large collections of data are normally stored with the use of a relational database management system (DBMS), for example marketing databases are used in marketing analysis, yet most data mining technology is not aware of the DBMS making data mining difficult to apply.

Dunham (2003) suggests that the complexity of the data mining application is similar to the complexity of databases in the 1960s - then application programmers had to create an entire database environment each time they wrote a program - and data mining may evolve in the next decades in a similar way database technology has evolved to become easy to use and develop. Research in the areas of data mining and databases identified the significance of integrating data mining with a DBMS in making data mining a technology available to the

non-expert user (Deogun et al., 1997; Holsheimer et al., 1995; Netz et al., 2000).

This chapter presents a discussion of the reasons that have motivated integrating data mining with a DBMS (section 6.1). The chapter then discusses a number of considerations when integrating data mining with a DBMS (section 6.2). The chapter continuous with a discussion of different approaches followed in integrating data mining with a DBMS (section 6.3). The chapter ends with a discussion of the reasons that integrating clustering with a DBMS has received less attention (section 6.4).

6.1 Reasons for Integrating Data Mining with a DBMS

Research that has looked at integrating data mining with databases has been motivated by a number of reasons including:

- Simplifying the *Knowledge Discovery in Databases* (KDD) process.
- Achieving main memory independence.
- Supporting on-line data mining.

6.1.1 Simplifying the KDD Process

Data mining is part of a larger and more complex process known as Knowledge Discovery in Databases (KDD) (Béjar et al., 1997; Michalski & Kaufman, 1998). The CRoss-Industry Standard Process for Data Mining (CRISP-DM) (Chapman et al., 1999; Harper & Pickett, 2006; Shearer, 2000) model reflects the complexity

of the overall KDD process (see figure 6.1). As the model shows, a number of subprocesses have to be completed and often repeated before a useful data mining result is produced. The database appears in the middle of the model as it plays an important role in all the subprocesses of KDD.

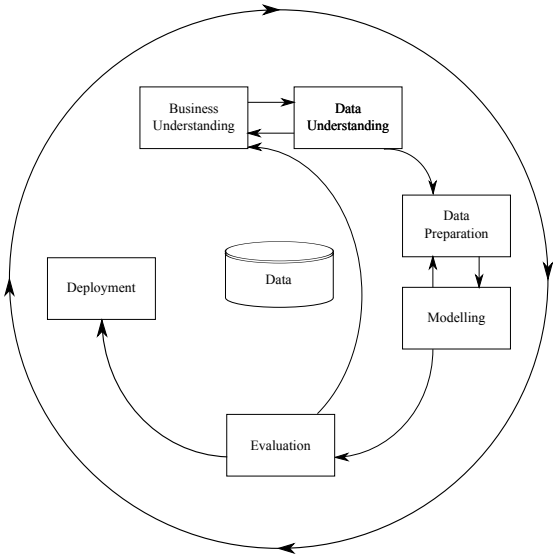

Figure 6.1: Cross-Industry Standard Process for Data Mining (Shearer, 2000)

A fuller integration of data mining and databases could improve the overall KDD process. For example, by running data mining and storing the results in the DBMS, the repetitiveness of KDD could become more manageable. This is because all the data and the data structure are available in each stage of KDD. At the *evaluation* stage (see figure 6.1), for instance, the user could employ the data dictionary or other data in the DBMS to reach better conclusions.

Part of CRISP-DM is the *data understanding* and *data preparation* stages. These two stages are considered quite expensive because of the repetition they involve (Brackman & Anand, 1996; Fayyad et al., 1996; Frawley et al., 1992). The two stages have become even more complex as the data sets that must be

extracted from a DBMS are larger. Chaudhuri (1998), Lu et al. (1996) and Freitas & Lavington (1996) are examples of work where the motivation has been to make the *data understanding* and *data preparation* stages simpler. Their work made use of SQL to identify data more relevant to a data mining problem, to summarise data for data mining or to make data available for data mining.

6.1.2 Achieving Main Memory Independence

Many data mining algorithms are main memory dependent (Boley, 2001; Dougherty et al., 1995; Jonyer et al., 2002; McCallum et al., 2000). As data mining is applied to larger and more complex data sets, it is important to remove main memory limitations on the performance of the algorithms.

Main memory limitations can be removed by mapping the complex calculations of an algorithm to SQL operations. Work in this area often follows the classification data mining technique (Sousa et al., 1998). Classification involves a training set and a testing set. The algorithm is applied on the training set, which contains a number of attributes. One of the attributes is considered to be the *class* that labels the data and guides the learning process. The resulting model is tested using the test data. An example of work that translates the classification problem into a DBMS problem by mapping the algorithm's processes into SQL, and achieves in this way main memory independence, is MIND (Wang et al., 1998). MIND is discussed in section 6.3.3.

6.1.3 Supporting On-line Data Mining

On-line data mining is defined in this thesis as the ability to mine data as it reaches the DBMS and provide a solution while the algorithm is running (Berkhin, 2002; Bradley et al., 1998).

Traditionally, the data mining task is performed off-line. Dunham (2003) refers to this as static data mining. Static data mining algorithms require a data set to be extracted from the DBMS and presented to the algorithm. The problem with this approach is that it is difficult to incorporate new tuples into the resulting model, particularly, with operational databases where data updates are common.

Dragut & Nichitiu (2004) offers the advantage of analysing data in real time. They have identified certain characteristics of an algorithm that support on-line data mining. For example,

- the ability to update an existing data mining model, and
- the ability to build a model requiring every tuple to be scanned and processed once.

Achieving on-line data mining through an integration with the DBMS has the advantage that some data mining solution can be made available to the user during running time. The DBMS offers the advantage of storing the data mining result alongside the data in the database as relation(s) that can be accessed using standard query tools. So while the algorithm is running some output is accessible and ready to be evaluated by the user. The problem with Dragut & Nichitiu (2004) is that a data mining model is only made available when the data mining process ends.

There are algorithms (Ordonez, 2006; Sousa et al., 1998; Wang et al., 1998) that can perform on-line data mining and present some solution to the user at run time. These algorithms have a better approach to on-line data mining because they are integrated with the DBMS. However, these algorithms require excessive disk accesses that limit their scalability.

6.2 Considerations when Integrating an Algorithm with a DBMS

When integrating data mining with a DBMS there are a number of issues that should be considered. These issues relate to two areas: (i) the data, and (ii) the resources.

6.2.1 The Data

Data stored in a DBMS is complex. Two aspects of this complexity are: (i) dynamic data, and (ii) structured data.

Dynamic Data: Data in a database is regularly updated. DBMS algorithms like indexing algorithms have the ability to update the underlying structure when new tuples arrive (Deschler & Rundensteiner, 2001; Katayama & Satoh, 1997; Sellis et al., 1987). Unfortunately, data mining algorithms often assume that the data set is static. By and large, to incorporate new data into an existing data mining model, a new data set must be extracted, which involves going back to the database, incorporating the new data into the data set and building a new model. To work with dynamic data an algorithm should allow an existing model

to be updated.

A classification of data mining algorithms, already discussed in this thesis in relation to clustering (see chapter 2), splits them into incremental and non-incremental, based on their ability to update an existing model (Lebowitz, 1987; Roure & Talavera, 1998; Sison & Shimura, 1996; Talavera, 2000; Weng et al., 2003). Incremental algorithms allow incorporation of new tuples into an existing model, whereas non-incremental algorithms can only incorporate new cases if a new model is constructed.

Lebowitz (1987) recognises that real world data, like most data stored in a database, is regularly updated and suggests that learning from real world data needs to be incremental. The reason being that with real world data the whole input data set is not necessarily known in advance. New data may be made available during or after the completion of the learning process. Incremental learning overcomes this problem by requiring every tuple to be processed without revaluating all the data and offers fast updating of an existing model. Therefore, algorithms that support incremental learning are better suited to the analysis of real world data.

Structured Data: Data mining algorithms normally expect the data to exist in the form of a flat file. Data stored in a DBMS tends to be structured as it follows a certain database model, whereas a flat file shows no structural relationships.

With the most popular database model being the relational model (Date, 1995; Elmasri & Navathe, 2000), a number of algorithms have been developed for mining relational databases. One example proposes interpreting simple relationships between relations to discover new knowledge about the database (Ketterlin

et al., 1995). The authors set out a method for using the entity relationship model (Connolly et al., 1996; Date, 1995) to discover knowledge at different levels of abstraction based on *one-to-many (1:M)* relationships. Clusters are built based on the instances in the many entity (components). Clusters are also built based on the instances in the 1 entity (composites). The second set of clusters is then expressed in terms of clusters of components supporting, in this way, discovery at several levels of abstraction simultaneously.

The work of Ketterlin et al. (1995) is using the Cobweb algorithm. They recognised that data stored in a database is structured following an entity relationship design. They also recognised that the entity relationship model is more expressive than a flat file and extended Cobweb to use the entity relationship model in its learning process. In doing so, they achieved mining structured data (structural learning) using an unsupervised learning algorithm [1] when most of the structural learning was done using supervised learning algorithms [2].

Another example proposes integrating knowledge resulting from analysing individual relations (Ribeiro et al., 1995). Their work uses the structure in the data to support data mining across multiple relations. A number of related relations are mined individually first and then the collective results are integrated and analysed further to discover knowledge that is based on all the relations.

All the examples discussed in this section have a common limitation. Although, the examples are proposing a method for applying data mining to relational databases as opposed to a flat file, they still require the data to be separated

[1] Cobweb is described as an unsupervised learning algorithm because it does not require knowledge of a class.

[2] Classification is described as supervised learning as it requires data to be labeled according to a class.

from the database as they rely on algorithms that are not DBMS aware. One of the aims of this thesis is to make conceptual clustering applicable directly to data that exists in a DBMS (see section 1). This will benefit the work of Ketterlin et al. (1995), in particular, as Ketterlin et al. (1995) proposed a system for clustering structured data that is based on Cobweb but their clustering system requires the data to be extracted from the DBMS. The work of Ketterlin et al. (1995) can also benefit from the fact that we scaled Cobweb to large data sets. In their paper, they mention that their clustering system would build better clusters if applied to large data sets.

6.2.2 The Resources

In-memory algorithms tend to be faster as disk access is slower than main memory access. When a DBMS is used, it is therefore important to maximise main memory usage and minimise database storage. An approach where the algorithm heavily relies on the DBMS even when there is adequate main memory (Sousa et al., 1998; Wang et al., 1998) may not be as good as one that allows the algorithm to adapt to the resources and only use the DBMS when it is absolutely necessary (Shafer et al., 1996).

An approach that adapts to the available resources and only uses the DBMS if it runs out of main memory must consider the I/O transfers between main memory and DBMS. The number of I/O transfers required relates to (i) the algorithm's data structure, and (ii) the evaluation it performs. This is because these two things determine the algorithm's ability to:

- exclude parts of data from main memory,

- keep in main memory the most requested data, and
- complete the analysis process with one disk scan of the raw data.

When making use of the DBMS to overcome main memory limitations, it is interesting to look at how the DBMS itself manages data. A DBMS works constantly between main memory and disk. The task of transferring data between the two environments is expensive. However, the DBMS reduces the cost of transferring data using various techniques such as (i) transferring the data in units of blocks to reduce the number of disk reads, and (ii) using caching to keep the most requested data in main memory. These techniques allow a DBMS to reduce the I/O cost, scale well to large data sets and be adaptable to changing resources.

6.3 Approaches to Integrating Data Mining with a DBMS

Sarawagi et al. (1998) defines five levels of integration between a data mining algorithm and a DBMS (Sarawagi et al., 1998, 2000):

1. loose-coupling through a SQL cursor interface or ODBC [1],
2. encapsulation of a mining algorithm in a stored procedure, [2]

[1] Open Database Connectivity (ODBC) provides a standardised API for accessing different DBMSs (Ramakrishnan, 2003).

[2] A stored procedure is a program stored in the database data dictionary available to all database applications to use (Koch & Loney, 1995).

3. caching the data to a file system on-the fly and mining,[1]

4. tight coupling using primarily user-defined functions and

5. SQL implementations for processing in the DBMS.

The levels of integration vary from loose integration, where the algorithm gets the data from the DBMS directly but the DBMS is treated as a data repository (Rajamani & Cox, 1999), to tight coupling integration, where part or all of the data processing is mapped to SQL and executed in the DBMS (see figure 6.2).

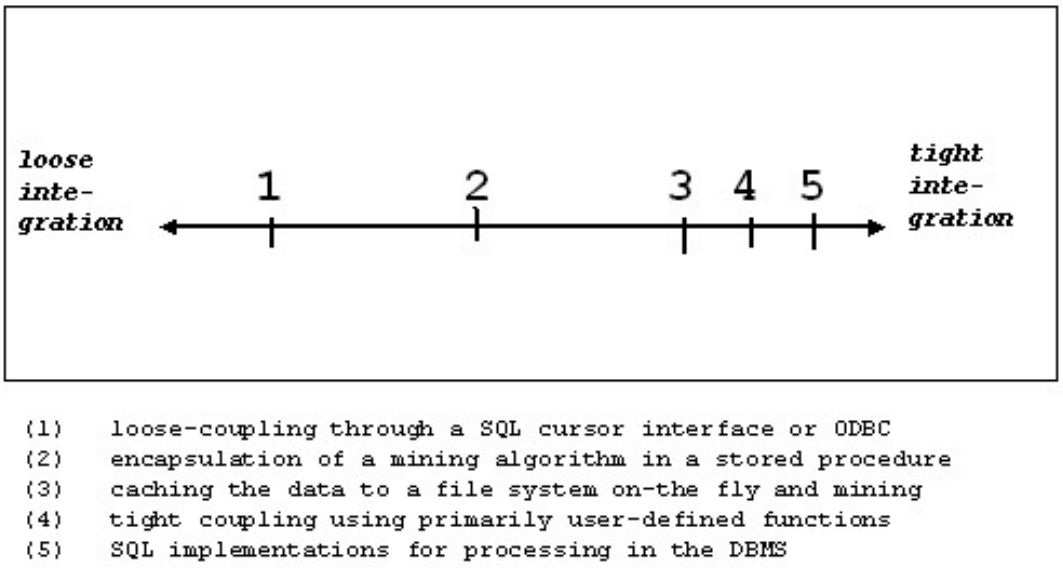

Figure 6.2: Sarawagi's Alternatives to Integrating a Data Mining Algorithm with a DBMS

Although, Sarawagis's levels of integration are good for understanding the degree of integration between data mining and databases, his classification does

[1] A cache is a temporary storage used to speed up the retrieval of frequently accessed data. Caching is a standard technique used in a DBMS to improve upon data access (Anton et al., 2002).

not lend itself well to understanding the type of changes in the underlying algorithm to achieve the integration. A different classification of the work that looked at integration is the following:-

- Integration that treats the DBMS as a repository.
- Integration that involves changing the DBMS.
- Integration that maps data mining to SQL.

The first approach involves no changes to the algorithm or the DBMS and is the same as Sarawagi's loose-coupling through a SQL cursor interface or ODBC. The second approach aims to support data mining by changing or enhancing the DBMS environment. The final approach involves changing an algorithm in order to make it SQL or DBMS aware.

6.3.1 Treating the DBMS as a Repository

This is the simplest and first type of integration tried in the literature. It is a loose connection between data mining and databases and normally uses an application programming interface (API) such as the Open Database Connectivity (ODBC). This type of connection with the DBMS has been employed by many data mining systems and algorithms, all using an API to read the data from the DBMS and make it available to the algorithm. Clementine (Clementine, 2005) and WEKA (Witten & Frank, 2005) are some examples of systems that use this type of connection.

The main advantage of using an API is that the system or algorithm has access to a wide range of DBMSs. For example, Clementine can extract data

from DB2 and Oracle without requiring any change in the application level or the DBMS. Another advantage is that this approach is very simple to implement because it only uses standard SQL to extract data from the DBMS. In addition, it provides the ability to access multiple databases and bring together data from different sources at the application level. Despite all the advantages, however, the use of an API does not allow an algorithm to take full advantage of the DBMS as discussed in the next section (section 6.3.2).

The algorithm treats the DBMS as a repository from where it retrieves the data. Once the data is read the entire process runs in main memory.

6.3.2 Changing the DBMS

This approach aims to support a range of data mining algorithms by implementing a common set of data mining operations in the DBMS (Clear et al., 1999; Geist & Sattler, 2002; Gray et al., 1997).

Sattler & Dunemann (2001) introduced primitives for supporting decision tree classifiers. These primitives are intended to serve a range of classification algorithms. Their approach looked at common functions between classification algorithms and developed primitives that support these functions, for example, the computation of the measure, gini-index (Wang et al., 1998).

Freitas & Lavington (1996) introduced primitives to support classification algorithms. They speed up the KDD process by reducing the interaction between the algorithm and the database. For example, their *Count by Group* primitive counts the number of tuples in each group of tuples with the same value (for the Group By attributes) using group by SQL statements.

6.3.3 Mapping Data Mining to SQL and Database Relations

This approach maps data mining operations to SQL and/or the data mining model to database relations. MIND (Wang et al., 1998) is an example of this approach. MIND builds a decision tree based on a training set (see figure 6.3) by operating in a DBMS.

Training Set

salary	age	credit rating
65K	30	Safe
15K	23	Risky
75K	40	Safe
15K	28	Risky
100K	55	Safe
60K	45	Safe
62K	30	Risky

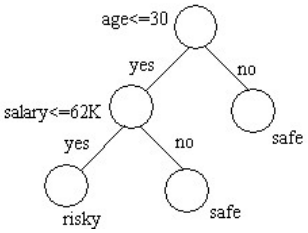

Figure 6.3: Example of Decision Tree

A leaf node in the decision tree (see figure 6.3) represents a class name, whereas an internal node is a decision node that represents an attribute value test that determines the path to the next level of the tree. To grow the decision tree, MIND has to decide the attribute value that best splits the data and best predicts the class. To decide the best split point at a node, MIND computes the *gini* index (Wang et al., 1998). The attribute containing the split point achieving the smallest *gini* index value is selected to be split the node.

MIND uses database relations in the process of deciding the best split point. It queries the relation that contains the training set and collects statistics about the data. It then uses the statistics to compute the *gini* index. The main database relations that MIND uses are:

1. A relation that stores the training set that is called $DETAIL$. A tuple in the $DETAIL$ relation is an example of a known class: $DETAIL(attr_1, attr_2, \ldots, attr_d, class, leaf\_num)$, where $attr_i$ is the ith attribute, for $1 \leq i \leq d$ and $class$ is the classifying attribute. $leaf\_num$ is the leaf node in the decision tree that a tuple belongs to.

2. A set of relations that store statistics necessary to compute the *gini* index, Dim_i. The Dim_i set of relations store attribute counts against a $class$: $Dim_i(leaf\_num, class, attr_i, count)$. $leaf\_num$ is a leaf node in the decision tree that represents a class. A Dim_i relation is created for every attribute in the training set.

3. A relation that stores best split values and the corresponding *gini* index measurements for each leaf node in the tree with respect to $attr_i$: $MIN\_GINI(leaf\_num, attr_i, attr\_value, gini)$.

MIND queries the $DETAIL$ relation to insert the necessary statistics in the Dim_i relations. An example of that query is shown in figure 6.4. After populating the Dim_i relations, the algorithm uses the statistics in the Dim_i relations and computes the *gini* index for each leaf node at each possible split value of the $attr_i$. The computation results are inserted in the MIN_GINI relation.

The MIN_GINI relation contains the best split value and the corresponding *gini* index measurement for each leaf node in the tree with respect to the

```
INSERT INTO DIM_i
SELECT leaf_num, class, attr_i, COUNT(*)
FROM DETAIL
WHERE leaf_num <> STOP
GROUP BY leaf_num, class, attr_i
```

Figure 6.4: MIND: Data Summarisation

```
CREATE VIEW BEST_SPLIT (leaf_num, attri, attr_value)
AS
SELECT leaf_num, attri, attr_value
FROM MIN_GINI a
WHERE a.gini = (SELECT MIN(gini)
FROM MIN_GINI b
WHERE a.leaf_num=b.leaf_num)
```

Figure 6.5: MIND: Best Split

$attr_i$. The algorithm queries the MIN_GINI table and produces views [1] that help it decide the best split point for a leaf node. An example of a view is the $BEST\_SPLIT$ view (see figure 6.5), which shows the overall best split value for each leaf node: $BEST\_SPLIT(leaf\_num, attr\_name, attr\_value)$.

MIND has the advantage of requiring very little data in main memory as most data used in the decision tree growing process is stored in the database. There is only one data structure that the algorithm requires in main memory and that represents the growing decision tree. The data structure represents a node with a unique number and uses little main memory space. The disadvantage of MIND's approach is the execution of a large number of SQL statements. Most data mining algorithms use SQL to read the raw data from the database (Mehta et al., 1996). MIND, on the other hand, executes SQL statements at different

[1] A view is a stored SQL statement that the DBMS user can treat as another table (Koch & Loney, 1995).

steps in the analysis process and often the same SQL statement many times. For example, the query in figure 6.4 is executed many times in order to collect the necessary statistics for all the attributes in the $DETAIL$ relation.

MIND achieves scalability to large data sets by using the DBMS's parallel processing. The parallel DBMS supports scalability through concurrent executions of SQL. For example, the queries that populate the Dim_i relations above can happen in parallel. Parallelisation of the SQL may achieve scalability but a parallel DBMS implementation may not be available or it may be considered expensive. Furthermore, it is an approach to scaling data mining algorithms that can be described as orthogonal to other approaches to scalability. For example, the work in Shafer et al. (1996) achieves scalability through the use of efficient data structures. Their work can immediately benefit from a parallel DBMS implementation without any changes to the algorithm they present.

Classification involves a relatively small numbers of operations. Every evaluation that the algorithm carries out is performed with regard to a known class. Clustering, on the other hand, considers different classes in the data as it looks for a natural classification. As a result, clustering has to perform more computations and is a more difficult to map to SQL. For this reason, most examples of data mining and DBMS integration do not use clustering.

The work of Ordonez (2006) is an example of a clustering algorithm that is integrated with a DBMS. The integration approach that Ordonez (2006) follows is similar to that of MIND as it employs a number of tables to summarise the data, which are then used to perform the clustering evaluation and follow a clustering strategy. Ordonez recognises that integrating data mining with a DBMS has many advantages but also that the extensive use of SQL queries can

have a negative effect on performance as SQL has high overhead. Ordonez's work achieves scalability by assuming that the number of the desired clusters is small. A better solution would reduce the number of the required SQL queries.

6.4 Discussion

The work in the area of integrating data mining with a DBMS has mainly focused on the data mining techniques of association and classification (Agrawal et al., 1996; Chaudhuri et al., 1999; Hipp et al., 2001; Lu, 2001). The reason that clustering has received less attention is related to its ability to scale.

Association and classification algorithms are normally less expensive because they involve a simpler analysis process than clustering. As we mentioned earlier in this thesis, classification uses an output field to guide the learning process. In clustering, where data is unlabelled, different alternatives have to be considered to form the best groups. As a result, clustering is often considered a more expensive data mining technique and is not favoured for an integration with the DBMS.

There are examples in the clustering area that prove that clustering algorithms can successfully scale to large data sets. An interesting example is BIRCH (Zhang et al., 1996), already discussed in chapter 2. BIRCH scales to large data sets and shows good performance with restricted resources. However, BIRCH has been developed for numeric data and scalability of numeric data is not considered as difficult as numeric data summarisations are more effective. Other examples of scalable clustering are the categorical data algorithms ROCK and LIMBO. Both algorithms have shown that categorical data clustering is scalable. However, from

a database perspective, ROCK and LIMBO have their disadvantages. They both involve different phases in their clustering approach and require user input that affects the quality of clustering (see chapter 2). CLIMIS, on the other hand, is a scalable clustering algorithm for categorical data with properties that suit a DBMS.

One of the aims of this thesis was to identify clustering algorithm properties that suit a DBMS (see chapter 1). The properties of CLIMIS that make it suitable to a DBMS are being incremental, non-parametric and involving a single phase in its clustering approach. All these properties make CLIMIS suitable to a DBMS because they require minimum user input and make the algorithm easy to use.

Chapter 7

Integrating CLIMIS with a DBMS

In this chapter, we discuss how we have extended CLIMIS to integrate the algorithm with a DBMS. The chapter starts with a discussion of a first attempt to integrate conceptual clustering with a DBMS and describes some of the lessons learned. The original approach had many advantages but the algorithm executed too many SQL statements and its performance was poor. CLIMIS has been designed to overcome this limitation.

7.1 The Cobweb/IDX Approach

The goal of this implementation was to improve the user's interaction with the data mining process. Cobweb/IDX (Lepinioti & McKearney, 2007, 2008) stores its clusters in standard database relations in a similar manner to Wang et al. (1998), discussed in chapter 5. The core of the Cobweb tree is stored in three

tables. First, the tree structure itself is stored as a two column table using a traditional parent/child hierarchical relationship, $cw\_tree(parent, child)$. The second table is the *cluster values* structure that describes the attribute/value probabilities for each cluster, $cw\_values(cluster, att, value, probability)$. Finally, the *cluster content* table, $cw\_cluster\_content(cluster\_identifier, primary\_key)$, describes the content of each cluster using a pair and links the cluster hierarchy to the original data. In addition to these three tables, there are a number of tables that improve the ease of use or performance of the algorithm.

Category utility is calculated using a series of aggregate queries. For example, the calculation of the sum of the squared conditional probabilities in cluster `10` uses the query:

```
select sum(probability*probability)
from cw_values
where cluster=10
```

One advantage of this approach is that the algorithm can be implemented using stored procedures that are optimized for use in the database management system. A second advantage of storing the clusters in relations is that the clusters can be queried using existing SQL interface tools.

Implementing the Operators

Implementing Cobweb in PL/SQL (*Procedural Language/Structured Query Language* (Koch & Loney, 1995)) allowed many of the algorithm's calculations to be performed as database queries within the database management system. For example, counting the number of attribute values across the data set can be

executed efficiently using an aggregate query and a full table scan. Similarly, counting a subset of attribute values can be performed equally efficiently using an indexed search.

The standard Cobweb algorithm has four operators as discussed in chapter 2. A single controlling procedure, `add_instance`, is responsible for iteratively stepping down the cluster tree and applying a *preview* procedure to evaluate each of the four operators and to select the *best operator* at each level. As the algorithm evaluates each operator, it changes the Cobweb data tables and assesses the quality of the resulting clusters. After each step, the changes to the data tables must be reversed. The changes are applied to a set of temporary tables and the tables are merged using standard SQL set operators. Standard database views are used to combine the contents of the main Cobweb tables and the temporary tables. For instance, the following is an example of one of the views:

```
CREATE OR REPLACE VIEW
   CW_TREE_PSEUDO_ALL (PARENT,CHILD)
AS
SELECT parent, child
   FROM cw_tree
UNION ALL
SELECT parent, child
   FROM cw_tree_pseudo
```

User Interface Design

The interface to Cobweb/IDX supports two processes: updating the clustered data set and predicting similar records using the cluster hierarchy. The update process monitors the indexed relation for new records and incorporates them into the cluster hierarchy. The prediction process takes a sample record and proposes similar records or missing values using the cluster hierarchy.

From the user's perspective, updating Cobweb/IDX was intended to be similar to updating existing database index structures. Typically index structures run in the background and are updated automatically when the data is changed. For example, the B+-Tree index (Knuth, 1973) is created using the CREATE INDEX command and requires no further intervention from the user. To achieve this level of integration, Cobweb/IDX is created or dropped using a script and the update process is triggered when new records are inserted into the indexed relation Ceri et al. (2000). This design also allows the update process to be separated from the data insertion process and helps to improve overall performance. The architecture of the update process is shown in figure 7.1.

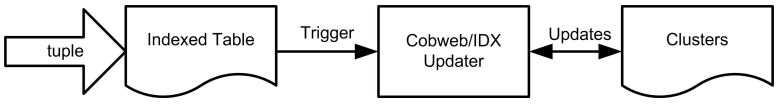

Figure 7.1: Cobweb/IDX Update Process

Predicting missing values using Cobweb/IDX is modeled on the *Query By Example* (Zloof, 1975) approach to querying relational databases.

The prediction process uses two tables: input and output. The input table is empty except when the user inserts (incomplete) records into it. These records are removed from the input during the prediction process. When Cobweb/IDX

reads a new input record, it uses the *predict* operator to process the record and identify any missing values. At present, missing values are indicated by *null* values in the input record. The predict operator proposes values for the null attributes based on the other values in the identified cluster. The output table contains the input record but with the missing values replaced with the value predicted by the Cobweb/IDX index. At present, the index predicts one value for each null but could be adapted to predict more than one value with an appropriate probability. The architecture of the search process is shown in figure 7.2.

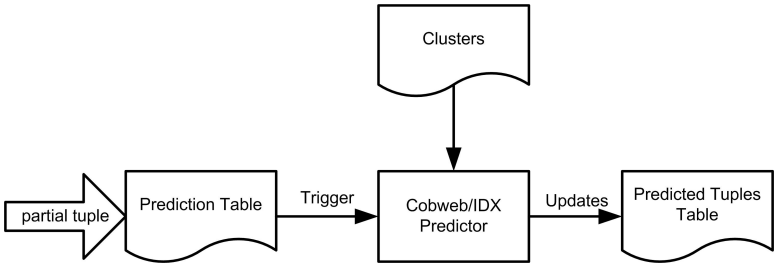

Figure 7.2: Cobweb/IDX Search Process

This input/output structure provides a convenient method of using the index and fits well with existing database query tools. For example, the input/output tables can be directly linked into a Microsoft Access database through the *linked table* facility and a convenient form-based interface built on top for non-technical users. Predicting missing values is simply a matter of inserting records into the input table and reading the results as they appear in the output table.

Advantages/Disadvantages

The original implementation of Cobweb in a DBMS had many advantages:

- It supported simple data mining. Data in the database could be indexed

and clusters queried using standard SQL. In this way, the running of the algorithm was hidden from the a user.

- The DBMS maintained the data structures without user intervention.
- It provided logical and physical independence as it was possible to add or drop the Cobweb/IDX index without affecting any other DBMS structure or operation.
- It achieved memory independence. As the algorithm uses the DBMS for computing *category utility* it requires very little data in main memory.
- It was based on a good incremental algorithm that did not suffer from ordering effects in the data.

The implementation of Cobweb/IDX was important, mainly, because it supported a simple application of data mining from a user's point of view. However, the algorithm had some disadvantages that must be considered:

- It had to perform a large number of complex queries to cluster a tuple, which affected the performance of the algorithm. For example, to calculate the CU and evaluate a clustering, one of the queries the algorithm had to perform was the aggregate query on the *cw_values* relation described above. This query was executed to evaluate:
  - the *incorporate* operator for each cluster,
  - the *split* operator for the best cluster, and
  - the *merge* operator for the best and second best clusters.

- It made insufficient use of the available resources. The algorithm made limited use of main memory even when the entire data set could fit in memory.

7.2 The CLIMIS Approach

The CLIMIS approach achieves a looser integration with a DBMS and avoids the disadvantages of Cobweb/IDX by using a cache (Stierhoff & Davis, 1998). It implements the two data structures that CLIMIS uses, *CLIMIS tree* and *PD-structure* (both discussed in chapter 3), as a cache. A cache is a temporary storage used to speed up the retrieval of frequently accessed data (Anton et al., 2002). By using a cache, CLIMIS reduces the number of queries executed and makes efficient use of main memory.

7.2.1 CLIMIS Tree

We will refer to a tree built by the CLIMIS algorithm as the *conceptual clustering tree*. The *conceptual clustering tree* is different to the *CLIMIS tree*. The *CLIMIS tree* stores all or part of the *conceptual clustering tree* depending on the amount of main memory available. If the algorithm runs out on main memory, the *conceptual clustering tree* exists in the *CLIMIS tree* and in the database (see figure 7.3).

The size of the *CLIMIS tree* is controlled by a user specified parameter. This can change in the future as the size of the cache could be proportional to the amount of memory available to the algorithm. The parameter controls the maximum number of tree objects that are allowed in the *CLIMIS tree*. If the maximum number of tree objects is exceeded, the algorithm stores clusters in the

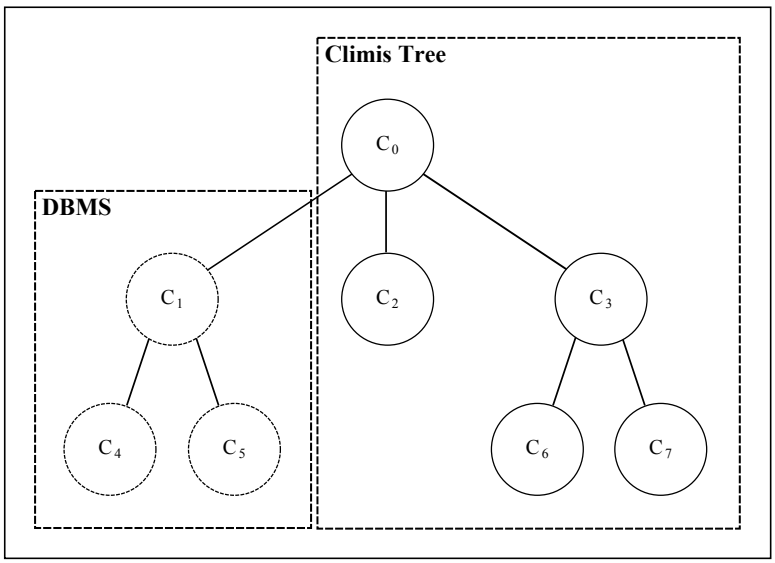

Figure 7.3: Conceptual Clustering Tree

database.

Figure 7.4 shows how the *CLIMIS tree* is implemented as a cache. The example is based on the tree shown in figure 7.3. The cache stores two types of objects: *parent objects* and *child objects*. A *parent object* represents a cluster in the *CLIMIS tree* that has children. A *parent object* represents a cluster with its unique identifier and points to a list that stores all of the cluster's children, *child list*.

A *child object* represents a cluster in the tree with its *cluster summarisation (CS)*. All *parent objects* are stored in one list, *parent list*. The algorithm adds new *parent objects* at the beginning of the list and removes objects from the end of the list. Similarly, the algorithm adds a *child object* at the beginning of the list and removes *child objects* from the end of the list.

When a leaf cluster is expanded, the algorithm adds a new *parent object* in the *parent list* and two *child objects* in a new *child list*. When the *disjunct* operator

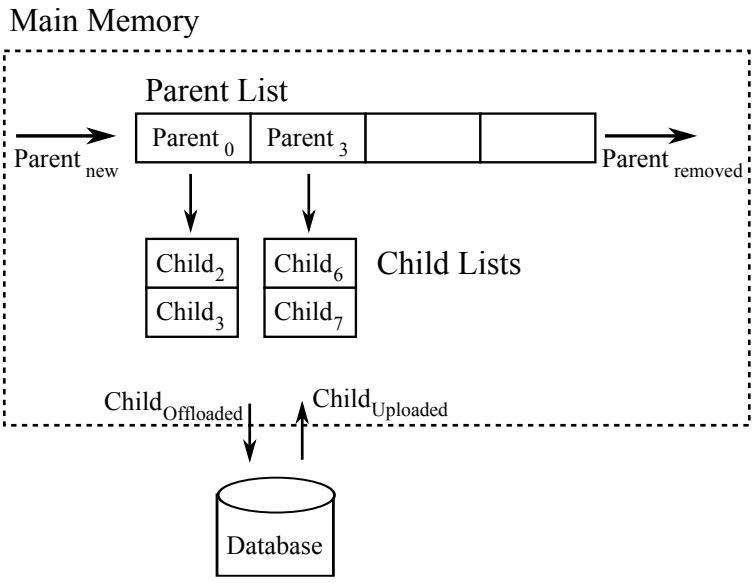

Figure 7.4: CLIMIS Tree as a Cache

is implemented, the algorithm adds a new object to the *child list* of the *parent object* representing the current cluster. When the *split* operator is implemented, the algorithm removes the split *parent object* from the *parent list* and merges its *child list* with the *child list* of the parent of the split cluster. When the *merge* operator is implemented, the algorithm removes two *parent objects*, merges their *child lists* and adds a new *parent object* to the *parent list*, which points to the merged *child lists*.

When a new object is added, parent or child, the algorithm checks to see if there is a free space in the cache. If there is a free space, the algorithm adds the new object in the cache. If there is no free space, the algorithm writes an object to the database and creates a free space. The algorithm always removes objects from the *child list* of the last *parent object* in the *parent list*. If all the children of the last *parent object* have been removed, then the algorithm creates a free space

by removing the last parent object.

A *parent object* requested by the algorithm is moved to the beginning of the *parent list*. In this way, the algorithm ensures that the removed object represents the least recently accessed cluster.

Before reading the children of a parent from the cache, the algorithm always checks a flag in the *parent object* to see if it has all its children in the cache. If the parent is missing children, the algorithm reads the missing children from the database.

7.2.2 Mapping the CLIMIS Tree to the Relational Data Model

CLIMIS uses two relations to store data in the database (see section 5.2.2):

1. The tree relation, $tree(Parent, Child, Type, Probability, SUMSQ)$.

2. The cluster content relation, $cluster\_content(Leaf, Recid)$.

A tuple is represented in the *cluster_content* relation and in main memory with its *rowid*, which is a unique identifier the DBMS provides for every tuple. Each time the *tree* relation is queried to get a leaf cluster, the *tree* relation is joined with the *cluster_content* relation to get the leaf cluster's tuples.

DBMS accesses: In the development of the *CLIMIS tree* cache, we aimed to use low cost queries to reduce the I/O traffic between the DBMS and the algorithm. Low cost means that the accesses relate to a small number of tables and they read or update only a small number of tuples. Queries that relate to the *CLIMIS tree* and read data from the DBMS are:

- Reading a parent with its children when a parent is not found in the cache.
- Reading the child of a parent when a cached parent is missing a child.

Both queries involve a join between the *tree* relation and the *cluster_content* relation to relate a cluster to its tuples:

```
select child, type, probability, sumsq_probability, recid
from tree, cluster_content
where tree.child = cluster_content.child
and parent =  current cluster
```

Other types of database accesses that relate to the *CLIMIS tree* include:-

1. Insert a child cluster into the *tree* relation to make space in the cache. If the cluster is a leaf it will also involve inserts into the *cluster_content* relation.
2. Delete a cluster that is removed by a split from the *tree* relation. This is never a leaf cluster and, therefore, the delete only involves the *tree* relation.
3. Update the *tree* relation to change the parent of children after a split.
4. Update the *tree* relation after a merge as merged clusters become children of the new cluster.

Indexing can be employed to support faster execution of the above queries, deletes and updates. This is because all these database accesses involve searching a relation (or two relations) against some criteria. For example, to delete a cluster that does not exist due to a split, the algorithm has to find the corresponding tuple in the *tree* relation and remove it. The only database access described

above that is not supported by an index is the insert but this is the simplest operation to perform in a database as new tuples are added in the end of the table. Also, most of the database accesses described above involve a single table. Database accesses that involve more than one table tend to be more complex as joining two tables together requires searching the tables to base the join on a primary, foreign key relationship.

Unlike the Cobweb/IDX implementation, which accesses the *tree* relation every time *category utility* is calculated, CLIMIS accesses to the *tree* relation via the cache and so it makes less requests to the database. The most often requested data exists in main memory as the experimental analysis of the *CLIMIS tree* hit rate will show in the next chapter.

7.2.3 Cache Replacement Policies

The most common cache replacement policies are: (i) least recently used (LRU), and (ii) least frequently used (LFU) (Gan & Suel, 2009; Jaragh & Hasswa, 2005; Selvakumar et al., 2004). LRU involves keeping track of the time an object is accessed. When the cache runs out of free space, the object that is replaced is the least recently accessed object (Gan & Suel, 2009; Lee et al., 2001; Venketesh et al., 2006). The LFU replacement policy involves keeping a counter for every object. The counter indicates the frequency of accesses to an object. The least frequently accessed object is the object that is replaced (Gan & Suel, 2009; Lee et al., 2001; Venketesh et al., 2006).

LFU does not have a mechanism to know if an object was accessed recently even if an object has high frequency of accesses. It is possible for an object to build

high frequency and then never be accessed again. However, because of the object's high frequency, this object will remain in the cache 'polluting' it (Selvakumar et al., 2004; Sokolinsky, 2004). This scenario is possible with CLIMIS. As CLIMIS is based on Cobweb, it has inherited many of Cobweb characteristics one of them being Cobweb's tendency to produce a skewed tree (Zhang, 1997). With a skewed tree, it is possible certain objects (clusters) to be accessed (visited) frequently at the beginning of the clustering process but never again later on. These objects (clusters) would 'pollute' the cache if the LFU policy was followed.

Considering the nature of CLIMIS, LRU was thought as a more appropriate replacement policy and is the policy followed in this work. The policy that was implemented is a variation of LRU. The objects in the cache are ordered according to the time they were accessed. The most recently accessed objects appear at the beginning of the list, whereas the least recently accessed objects appear at the end of the list (discussed in section 7.2.1). The object that is replaced is the least recently accessed object.

7.2.4 PD-structure

As discussed in chapter 3, the *PD-structure* stores conditional probabilities of attribute values in the leaf clusters. The *PD-structure* has been implemented as a cache to ensure that frequently requested attribute values exist in main memory.

In a database index, all accesses to the database happen through a common structure so there is never more than one copy of the same block in main memory. Similarly, CLIMIS uses the *PD-structure* to ensure that there is never more than one copy of a probability distribution in main memory. The *PD-structure*

stores all or part of the attribute value probabilities depending on main memory availability. When the cache stores part of the conditional probabilities then the rest of the conditional probabilities can be found in the database.

The *PD-structure* uses two lists - *AttVal list* and *cluster list* - and a matrix that shows the relationship between entries in the two lists. The *AttVal list* stores attribute value objects (*AttVal object*) that represent attribute values covered in the leaf clusters. The *cluster list* stores cluster objects. A *cluster object* represents a leaf cluster with its unique identifier.

When an attribute value appears in a leaf cluster for the first time, the algorithm relates the *cluster object* (C_k) to the *AttVal object* (V_j) by storing the conditional probability in the matrix (see figure 7.5).

p(V|C)	C_1	C_2	...	C_k
V_1	$p(V_1\|C_1)$	$p(V_1\|C_2)$		$p(V_1\|C_k)$
V_2	$p(V_2\|C_1)$	$p(V_2\|C_2)$		$p(V_2\|C_k)$
V_3	$p(V_3\|C_1)$	$p(V_3\|C_2)$		$p(V_3\|C_k)$
...				
V_j	$p(V_j\|C_1)$	$p(V_j\|C_2)$		$p(V_j\|C_k)$

Figure 7.5: PD-structure

Objects are added at the beginning of the *cluster list* and removed from the end of the *cluster list*. When a new leaf cluster is created the algorithm checks if there is a free space in the *cluster list*. If there is a free space the algorithm adds a new *cluster object* at the beginning of the *cluster list*. If there is no free space

the algorithm creates one by removing a *cluster object* from the end of the list. Similarly, objects are added at the beginning of the *AttVal list* and removed from the end of the *AttVal list*.

DBMS Accesses: The *PD-structure* is space bound. The size of the *cluster list* and *AttVal list* is controlled by a user specified parameter. This can change in the future as the size of the cache could be proportional to the amount of memory available to the algorithm. The parameter controls the maximum number of *cluster objects* and *AttVal objects* that are allowed in the *PD-structure*. If the maximum number of *AttVal objects* and *cluster objects* is exceeded, the algorithm removes an object.

When the *PD-structure* gets full, the algorithm checks if the information is in the *PD-structure* and if it is not, it updates the *PD-structure* using the DBMS. The accesses to the database that relate to the *PD-structure* are mainly reads from the original data set. The accesses to the database happen for the following reasons:

1. *To get the conditional probabilities of a leaf cluster*: This happens when the *PD-structure* is missing a requested leaf cluster.

2. *To get an attribute value's conditional probability in the leaf clusters*. This happens when the *PD-structure* is missing a requested attribute value.

The second query above is a lot more expensive than the first query as an *AttVal* object may relate to many clusters. The algorithm removes an *AttVal* object only after all the *cluster objects* have been removed from main memory.

```
Find the current parent cluster in the CLIMIS tree
    If parent cluster not found in the tree
        Look in the database
        Add parent with its children at the beginning of
        the CLIMIS tree
        Return the children of the current parent cluster
    Else if parent found but missing children
        Get the children that do not exist in the tree from
        the database
        Add children to the parent
        Return the children of the current parent cluster
    Else if parent found
        Return the children of the current parent cluster
End
```

Figure 7.6: Get Children

7.2.5 Interaction with the CLIMIS Algorithm

The CLIMIS algorithm relies on the *CLIMIS tree* cache and the *PD-structure* cache to compute the category utility and evaluate the operators. The algorithm updates the two structures, if appropriate, when one of the operators is implemented.

To evaluate the *incorporate*, *disjunct*, *split* and *merge* operators at a level, the algorithm reads the children of the current cluster from the *CLIMIS tree* once (see figure 7.6). When the algorithm identifies the *best operator*, it updates the *CLIMIS tree* and *PD-structure*:

- If *incorporate* is the *best operator*, the algorithm updates the *CLIMIS tree* and the *PD-structure* as shown in figure 7.7.
- If *disjunct* is the *best operator*, the algorithm updates the *CLIMIS tree* and

```
If incorporate = best operator
  Update the Cluster Summarisation of the best cluster in the CLIMIS tree
  If best cluster is a leaf
                Expand best cluster
                Create a new parent in the CLIMIS tree with two
                new leaf children
                Remove the old leaf cluster from the PD-structure
                Add the new leaf children in the PD-structure
  Else
                Move to the next level of the tree
  End
End
```

Figure 7.7: Implement Incorporate

```
If disjunct = best operator
              Add the new child in the CLIMIS tree
              as a child of the current cluster
              Add the new child in the PD-structure as a leaf cluster
End
```

Figure 7.8: Implement Disjunct

PD-structure as shown in figure 7.8.

- If *split* is the *best operator*, the algorithm updates the *CLIMIS tree* as shown in figure 7.9. Leaf clusters are never split and, therefore, the *split* operator does not update the *PD-structure*.

- If *merge* is the *best operator*, the algorithm updates the *CLIMIS tree* as shown in figure 7.10. Like *split*, the *merge* operator does not update the *PD-structure*.

The Cost of the Merge Operator: The *merge* operator is an expensive operator when running CLIMIS using the DBMS because of the number of

```
If split = best operator
        Update the Cluster Summarisation of the best cluster in
        the CLIMIS tree
        Add the children of the split cluster to the children of
        its parent in the CLIMIS tree
        Remove the split cluster with its children from the CLIMIS tree
End
```

Figure 7.9: Implement Split

```
If merge = best operator
        Add new parent for merge cluster with children best and second
        best clusters in the CLIMIS tree
        Remove best and second best clusters from current cluster child list
        Add merge cluster in the child list of current parent
End
```

Figure 7.10: Implement Merge

```
Get att val probability for merge.
    Go through the list of attribute values in the PD-structure.
    For every attribute value in the PD-structure,
    match the leaf clusters corresponding to the best and
    second best nodes to the leaf clusters in the PD-structure.
        If a leaf cluster not in the PD-structure check the database.
        If a leaf cluster is found, add 1 to the att val count.
    End
End
```

Figure 7.11: Getting Attribute Value Probabilities for Merge from the PD-structure

attribute value probabilities the algorithm has to compute for the merge *cluster summarisation*. The computation of every attribute value probability for the merge *cluster summarisation* may involve accesses to the database (see figure 7.11).

A possible way of solving the problem is avoiding the *merge* operator by replacing it with the *Not-Yet strategy* (Talavera & Roure, 1998). This is a strategy for improving the quality of incremental clustering. Talavera & Roure (1998) have compared the *Not-Yet strategy* with Cobweb's *merge* and *split* operators and found it to be more effective in correcting the ordering effect in the data. The strategy avoids including a tuple in a clustering if the tuple does not improve *category utility*. The tuple is kept aside and is clustered again when more tuples have been clustered. We would like to explore this strategy and use it in the CLIMIS implementation as it is expected to have a lower cost than *merge* and *split*.

7.2.6 Size of the Cache Data Structures

It is possible to calculate the maximum size of the *CLIMIS tree* when the algorithm runs entirely in main memory. We consider the worst case scenario, where the *cutoff* parameter is zero as this will produce singleton leaf clusters (see figure 7.12).

In the same way, the maximum size of the *PD-structure* can be calculated (see figure 7.13) if we assume that the size of the data set and the size of the attribute domains are known.

```
Size of Conceptual Tree = 2n-1 clusters

Size of Tree Relation = 2n-1 tuples

Size of Cluster_Content Relation =

Size of CLIMIS Tree = (2n-2) + (n-1)

(n = number of clustered tuples)
```

Figure 7.12: Size of the CLIMIS Tree

```
Size of Cluster List = n

Size of AttVal List  = av

Size of Matrix = n*av

(where, n = number of clustered tuples and
 av=number of attribute value pairs)
```

Figure 7.13: Size of the PD-structure

7.2.7 Full DBMS Interface

CLIMIS satisfies the criteria identified in Bradley et al. (1998) and Berkhin (2002) as important when integrating data mining in a DBMS: These criteria require that an algorithm:-

1. needs a maximum of one scan of the database,
2. utilises different scanning modes, if necessary (sequential, index, sample),
3. incorporates additional data incrementally,
4. processes every tuple once,
5. can be suspendable, stoppable and resumable,
6. can provide an on-line solution (some solution in progress should always be available).

CLIMIS reads data directly from the database so it can make use of the different scanning modes a DBMS offers. CLIMIS processes every tuple once when it builds the cluster summarisations. When new a tuple arrives, the summarisations can be adjusted without reference to previous tuples. This characteristic of CLIMIS is important as (i) the algorithm needs one scan of the database, and (ii) its interface can be implemented in a similar way to that of Cobweb/IDX (see section 7.1), which supports on-line data mining. Finally, the mapping of the CLIMIS data structure to relations means that CLIMIS can be implemented to be suspendable, stoppable and resumable.

Chapter 8

Evaluation of the CLIMIS Data Structures

This chapter presents an evaluation of the data structures that CLIMIS uses to interact with the DBMS:-

- The *CLIMIS tree*
- The *PD-structure*

According to Shatdal et al. (1994), a good cache has minimum cache misses and, therefore, maximum hit rate. The hit rate represents the number of successful calls to the cache out of the total number of calls made. The missed calls cause a disk access to retrieve data. We evaluate the cache structures *CLIMIS tree* and *PD-structure* by measuring percentage hit rate during data clustering. In the evaluation of the *CLIMIS tree*, the hit rate is shown against the ratio between the size of *CLIMIS tree* and the size of *Conceptual Clustering Tree*. In the evaluation of the *PD-structure*, the hit rate is shown against the ratio between

the size of the *PD-structure* and the total probability distributions at leaf cluster level.

8.1 CLIMIS Tree

Hit Rate

In this section, we evaluate the performance of the CLIMIS tree cache by measuring the hit rate - the number of successful calls out of the total number of calls made to the cache - with regard to the following:

1. the size of the CLIMIS tree cache,
2. the number of tuples in the data set,
3. the number of attributes in the data set, and
4. the type of the conceptual clustering tree produced by the CLIMIS algorithm.

With regard to the size of the cache, we applied the CLIMIS algorithm to the same data set a number of times while reducing the size of the cache. With regard to the number of tuples and attributes in the data set, we kept the size of the cache constant and applied the CLIMIS algorithm to different data sets with increasing number of tuples and attributes respectively. Finally, with regard to the type of conceptual clustering tree, we kept the size of the cache constant and applied the CLIMIS algorithm to two data sets, one producing a skewed and the other a balanced conceptual clustering tree.

The data sets used in the evaluation are real data sets and include the mushroom data set and subsets of the mushroom data set. We used zero *cutoff*, which meant that the CLIMIS algorithm built the maximum possible tree.

The experiment in figure 8.1, used different sizes of the *CLIMIS tree* so the ratio *CLIMIS tree cache/Conceptual tree* varied from 75% to 5%. 75% ratio means that the size of the *CLIMIS tree* allows only 75% of the clusters built by the algorithm to exist in main memory.

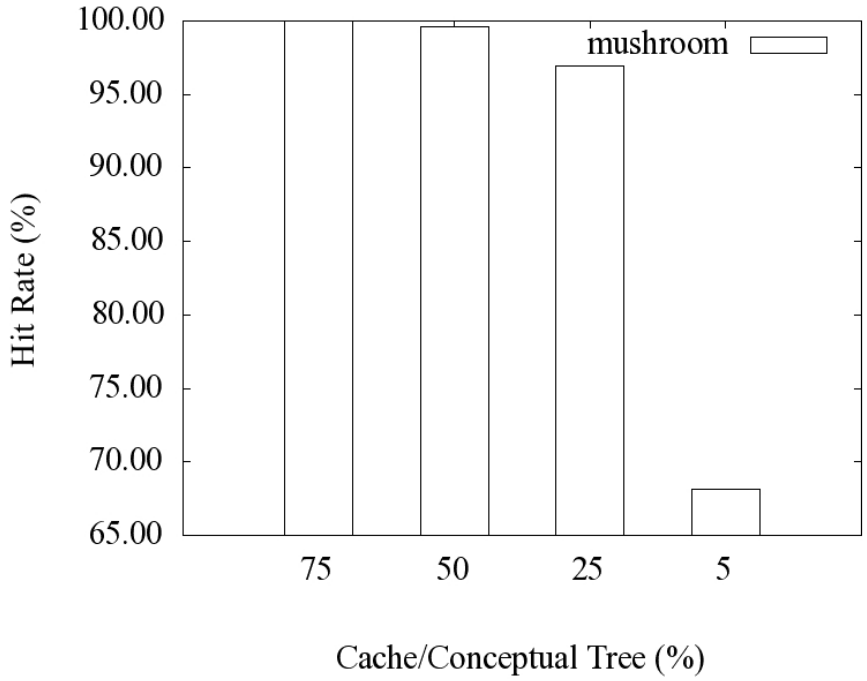

Figure 8.1: Hit Rate with Regard to Reducing Size of Cache

In all cases apart from one, the *CLIMIS tree* cache shows hit rate above 96% (see figure 8.1). Even when only a $\frac{1}{4}$ (25% ratio) of the conceptual tree fits in main memory, the hit rate is above 96%. This means that for more than 96% of the calls made to the cache the algorithm has found the required clusters in

main memory. The only time that the hit rate drops significantly is when the ratio is 5%. 5% ratio means that only a small portion of the *conceptual tree* fits in main memory. The hit rate drops to 68% because the cache fits very few of the clusters and must query the database to load required clusters.

All new clusters are stored in the cache. It takes time for a new cluster that is not important to become least recently accessed and be removed from the cache. With ratio of 5%, the algorithm does not have enough cache space to clear new non-important clusters and maintain a good number of older important clusters in the cache. As a result, the hit rate drops to 68%.

The same experiment was applied changing three other parameters:-

- Increasing the number of tuples.
- Increasing the number of attributes.
- Skewed versus balanced tree.

Doubling the number of tuples in the data set had little impact on the hit rate. As figure 8.2 shows the larger data set has a hit rate of 98% and 99%, which is similar to the hit rate of the smaller data set. The larger data set shows only slightly worse hit rate when 50% of the clusters fit in the cache (see figure 8.2). The difference in the hit rates can be explained by the larger number of singletons the algorithm produces with the large data set. Doubling the number of tuples, doubles the singletons created as every clustered tuple reaches a singleton. The more singletons the algorithm has to manage in the cache, the greater the chance important clusters will be discarded from main memory. The problem has been solved with the *cutoff* parameter, which reduces the number of singletons that CLIMIS produces.

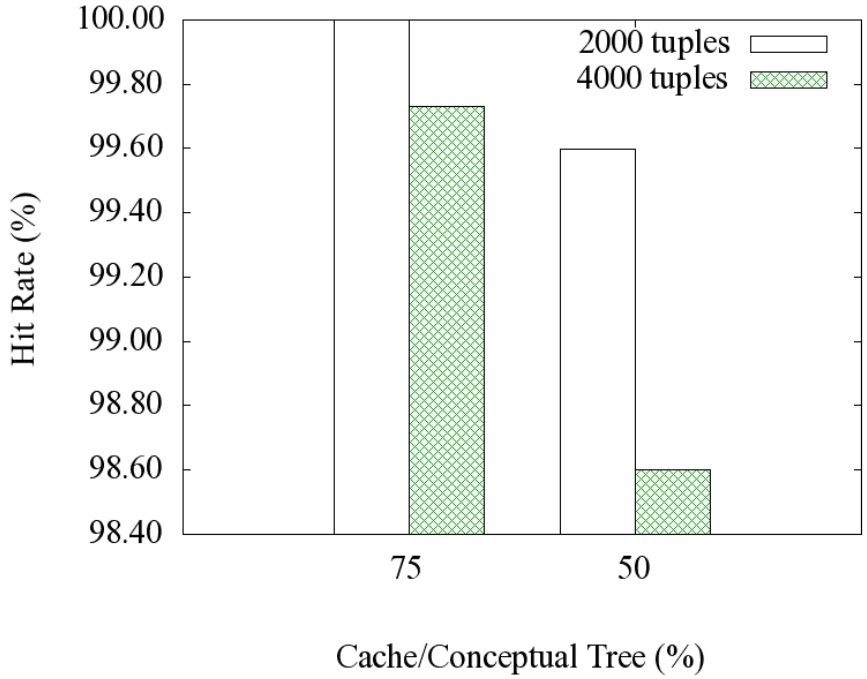

Figure 8.2: Hit Rate with Regard to Increasing No. of Tuples

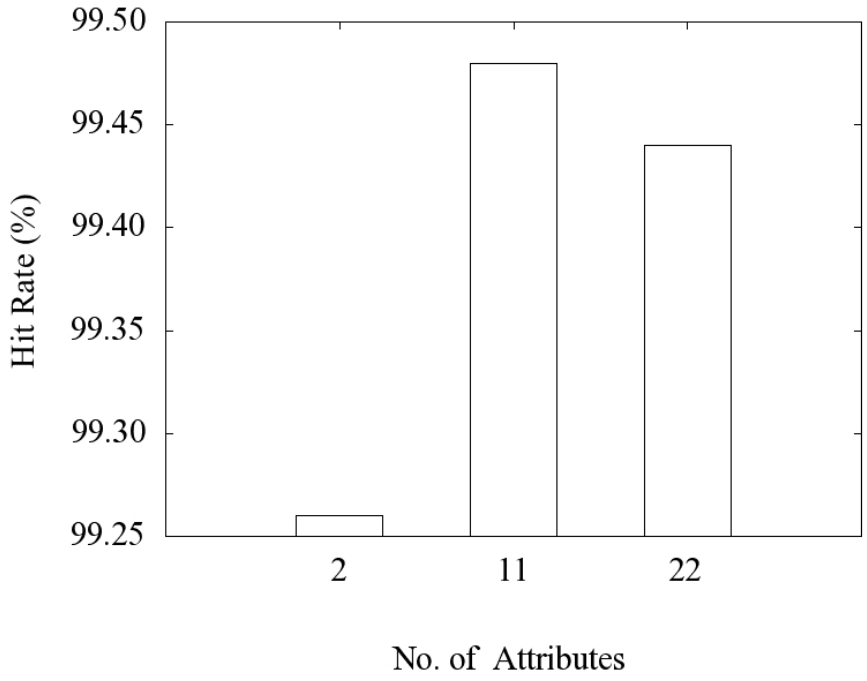

Figure 8.3: Hit Rate with Regard to Increasing No. of Attributes

Increasing the number of attributes had less impact on the hit rate than increasing the number of tuples. As shown in figure 8.3, increasing the number of attributes had little effect on the hit rate. In all cases, the hit rate is more than 99%. The increase in number of attributes has a smaller impact on the hit rate because more attributes in the data set do not result in a large increase in clusters. Unlike increasing the number of tuples, where every tuple causes the creation of a singleton, when $cutoff = 0$, adding an attribute to the data does not result in more singletons. It may even produce less singletons. It is interesting to see that 22 attributes produced better hit rate than 11 attributes. In this case, the 22 attribute data set has only $\frac{2}{3}$ of the clusters being singletons, whereas with 11 attributes, $\frac{4}{5}$ are singletons.

The final parameter that was varied in the evaluation of the *CLIMIS tree* cache was the data distribution. This was the only time a synthetic data set was used in this evaluation. The synthetic data set was produced with the data set generator discussed in chapter 4. We created a synthetic data set that had the same size as the mushroom data set but in comparison to the mushroom data set, it produced a balanced tree.

Figure 8.4 shows that the balanced tree has a lower hit rate than the skewed tree. As discussed in section 7.2.3 the *CLIMIS tree* follows the LRU replacement policy, which means that it replaces the least recently accessed object. When the algorithm builds a balanced tree and all the clusters are visited more or less equally, there is a lower chance that a cluster visited recently will be visited again soon. Therefore, there is less chance that a cluster (object) will be found in the cache next time it is requested. With a skewed tree, on the other hand, the hit rate is better because a recently visited cluster will most likely be visited again

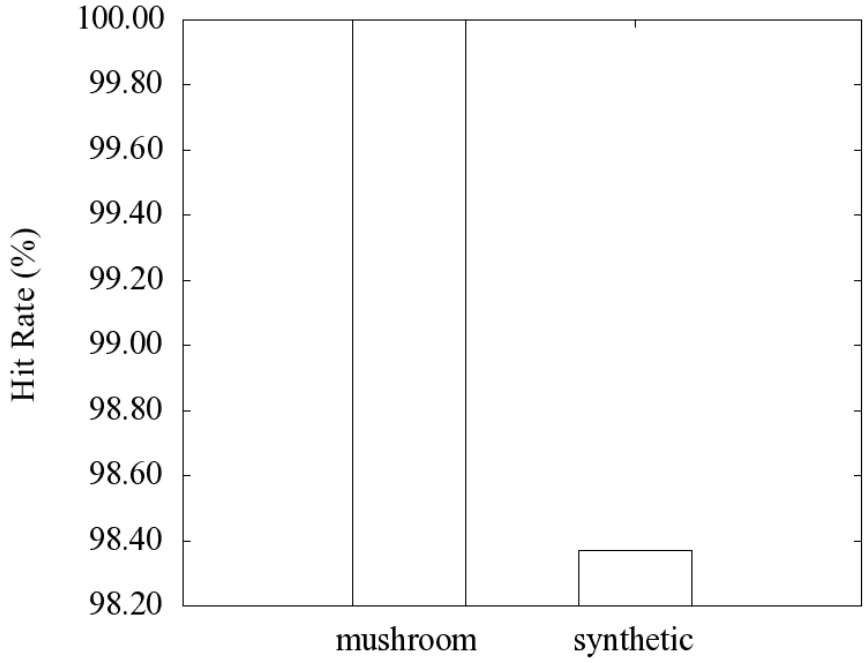

Figure 8.4: Skewed versus Balanced Tree

soon. Therefore there is a good chance it will exist in the cache when requested again.

Zhang (1997) criticised Cobweb (see section 2.7.1) for building a skewed tree of clusters as this is considered bad for performance. Part of her solution to the problem of scalability was to build a balanced tree so that when clustering a tuple, the path from root to a leaf is always the same. The same approach was also used in Andritsos (2004). Andritsos (2004) LIMBO algorithm builds a tree that is balanced for performance reasons. In this thesis, we recognise the skewed property of Cobweb, inherited by CLIMIS, as an advantage for performance as thanks to the skewed property there is a more efficient use of main memory. In addition, forcing the creation of a balanced tree may stop natural clusters from being discovered, which goes against the purpose of clustering. It has been shown that the way BIRCH (Zhang, 1997) builds a tree may stop natural clusters from being discovered (Han & Kamber, 2000; Sheikholeslami et al., 1998).

8.2 PD-structure

The size of data in the *PD-structure* is dependent upon the domain of the attributes in the data set. Unlike CACTUS Ganti et al. (1999), which assumes that the domain of a categorical data set can fit in main memory, in this thesis we avoided making this assumption as we wanted to achieve a memory independent algorithm. In this section, we evaluate the *PD-structure* with regard to hit rate. The data sets used in the evaluation are real data sets that include the mushroom data set and subsets of the mushroom data set.

Hit Rate

The factors considered in order to evaluate the *PD-structure* are:

- The size of the *PD-structure*.
- The number of tuples.
- The number of attributes.

A first look at figures 8.5, 8.6 and 8.7 indicates that varying the parameters - cache size, tuples and attributes - has a higher impact on the hit rate of the *PD-structure* compared to the *CLIMIS tree*. In all cases, the hit rate of the *PD-structure* is lower than 90%. This happens despite using exactly the same data sets in both cases.

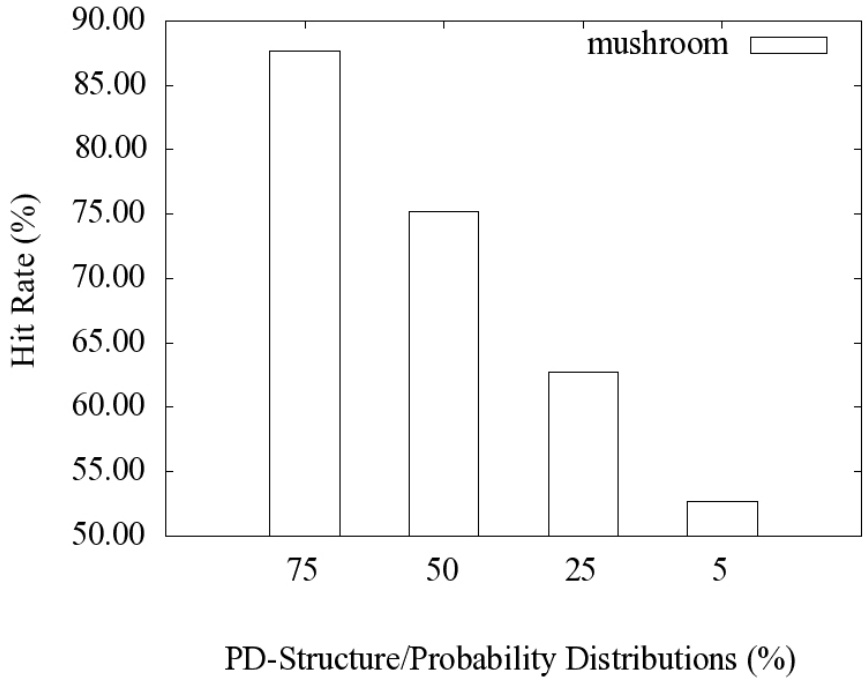

Figure 8.5: Hit Rate with Regard to Reducing Size of PD-structure

Figure 8.5 shows that as the size of the *PD-structure* is reduced, there is a quite large hit rate drop. The experiment was based on the worst-case scenario as every tuple in the algorithm stops at a singleton. A data set of 8,000 tuples produced 8,000 singleton clusters.

The effect that singletons have on the *PD-structure* relates to the number of leaf clusters that the *PD-structure* has to make available to the algorithm. When a new tuple is added in the tree, the algorithm has to find the leaf clusters that have values that match the new tuple. Many of the singleton clusters will match a new tuple. The *cutoff* parameter, discussed in chapter 4, reduces the clusters at leaf level and, therefore, the number of clusters the algorithm has to find in the *PD-structure*.

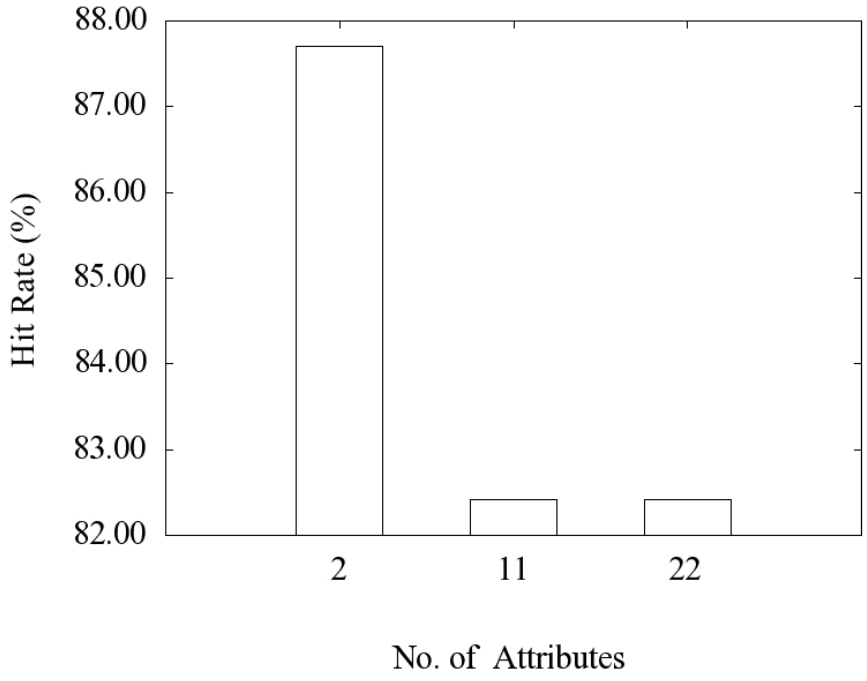

Figure 8.6: Hit Rate with Regard to Increasing No. of Attributes

Figure 8.6 illustrates the relationship between hit rate and increasing number of attributes. It is interesting that despite the difference in number of attributes, 11 attributes have more or less the same hit rate with 22 attributes. In this experiment, we keep the ratio [1] constant and vary the number of attributes. Because the ratio is the same for all experiment trials, the hit rate depends upon the probability distributions and how they are requested by the algorithm. More attributes does not necessarily mean a large increase in the probability distributions as that depends on the domains of the attribute values.

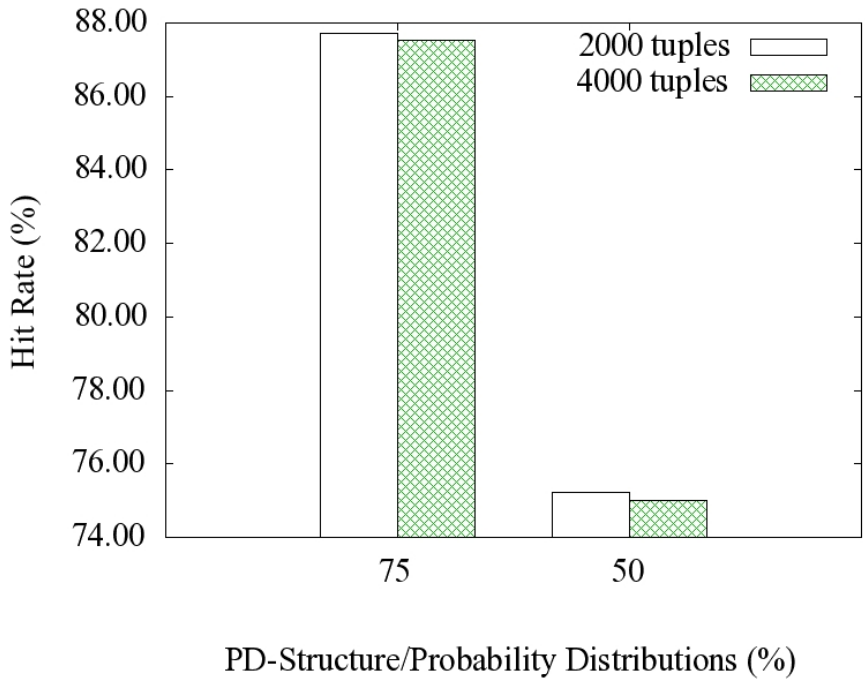

Figure 8.7: Hit Rate with Regard to Increasing No. of Tuples

Figure 8.7 shows the hit rate when increasing the number of tuples. Here, the hit rate of both data sets is almost identical when the ratio is between 75%

[1] The ratio between the size of the *PD-structure* and the total probability distributions at leaf cluster level.

and 50%. The reason is that the attribute domains are very similar for both data sets as they are subsets of the mushroom data. This means that increasing the number of tuples does not affect the type of clusters built. The result is a similar hit rate despite the difference in tuples.

8.3 Conclusion

The evaluation of the *CLIMIS tree* shows that the cache is effective even when its size is small in relation to the clusters produced. When the size of the cache is relatively good, an increase in number of tuples or number of attributes has little effect in the hit rate. This proves that CLIMIS makes efficient use of the main memory and does not require many SQL queries and updates to be performed each time a new tuple is clustered. The evaluation of the *CLIMIS tree* also shows that the cache performs better when the produced tree is skewed rather than balanced. This proves that the cache complements the nature of the CLIMIS algorithm.

The evaluation of the *PD-structure* shows that increasing the number of tuples or attributes has an impact on the hit rate but it may not be significant as it depends on the attribute domains and how they change as the tuples or attributes increase. The parameter that has the biggest effect on the hit rate is the size of the *PD-structure*. Although, the hit rate of the *PD-structure* is good, it is not as good as that of the *CLIMIS tree* when the size of the structure is reduced. This means that the algorithm requires more SQL queries to get the required probabilities. The hit rate recorded was down to the substantial number of singletons. Singletons can be easily avoided with CLIMIS, which proves that

the *PD-structure* is also an efficient structure and performs well even in the worst case scenario.

Chapter 9

Conclusions

One aim of this thesis was to investigate if it is possible to scale conceptual clustering to large data sets of categorical data. We introduced CLIMIS, a scalable conceptual clustering algorithm for categorical data. CLIMIS makes use of efficient data summarisations that allow the algorithm to scale conceptual clustering to large data sets. CLIMIS was evaluated against Cobweb and was found to have the same quality as Cobweb. Unlike Cobweb, however, CLIMIS can scale to large data sets. CLIMIS was also evaluated using comparable clustering algorithms developed for large data sets, LIMBO and ROCK, and was proven to have the same quality as both algorithms but better performance. In addition, compared to LIMBO and ROCK, CLIMIS is simpler to use.

Another aim of this thesis was to investigate if it is possible to have a scalable clustering algorithm with properties that suit data stored in a DBMS. CLIMIS is a scalable algorithm with properties that suit characteristics of data stored in a DBMS (data stored in a DBMS may be frequently updated, be of mixed data types and structured). CLIMIS is an algorithm that has been based on Cobweb and has maintained Cobweb's properties. Like Cobweb, CLIMIS is incremental and, therefore, applicable to frequently updated data. CLIMIS, also,

is extensible to mixed data and applicable to structured data as Cobweb has already been extended to mixed data and used for clustering structured data (Ketterlin et al., 1995).

Another aim of this thesis was to investigate if it is possible to make conceptual clustering main memory independent. CLIMIS was extended and a cache was used to integrate the algorithm with a DBMS and gain main memory independence. The cache was evaluated using hit rate analysis and was found having good hit rate even when the cache's size is small.

9.1 Contributions

This thesis contributes to the area of data mining and knowledge discovery in the DBMS in the following ways:-

- Proposes CLIMIS, a scalable conceptual clustering algorithm for categorical data (see section 3).
- Shows how conceptual clustering for categorical data can be scaled to large data sets by using data summarisations appropriate for categorical data (see section 3.2).
- Shows how conceptual clustering for categorical data can be scaled to large data sets without incurring any loss in the quality of the clustering results (see section 3.2 and 4.3).
- Shows how to achieve main memory independence by using a cache to interact with a relational DBMS (see section 7.2).

- Uses an approach to clustering that supports clustering of dynamic data (see section 2.7.2 and 7.2.7).
- Uses an approach to clustering that supports clustering of structured data (see section 2.7.2).
- Uses an approach to clustering that can be extended to mixed data and has immediate database applications (see section 2.7.2).

9.2 Limitations

A limitation of the CLIMIS algorithm is the cost of the *merge* operator when the algorithm uses the DBMS. To compute the *cluster summarisation* for the merge cluster, CLIMIS has to compute every attribute value probability for the merge cluster, which involves many calls to the cache. This makes the merge operator the most expensive operator of CLIMIS (see section 7.2.5). A possible way of solving the problem is avoiding the *merge* operator by replacing it with the *Not-Yet strategy*, which is an alternative to the *merge* operator for improving the quality of the conceptual clustering tree (Talavera & Roure, 1998).

9.3 Future Work

CLIMIS can be used to cluster large categorical data sets, it can work with limited resources and it can be applied to static or dynamic data that exists in a DBMS. Our main contribution to the community is scaling conceptual clustering of categorical data. We intend to build on our current research and extend CLIMIS to scale other existing algorithms.

9.3.1 Using CLIMIS to Scale ITERATE

The data structure used in CLIMIS can be used by algorithms such as ITERATE (Biswas et al., 1998), which are hierarchical and probability based. We intend to implement ITERATE using the CLIMIS data structures and show that ITERATE can scale to large data sets.

9.3.2 Extending CLIMIS to Relational Data Mining

There is an increasing interest in the area of relational data mining and evidence that interesting information can be learned from the relationships between database relations. Cobweb has already been extended to learn from structured data (Ketterlin et al., 1995; Lebowitz, 1987). Of particular interest is the work of Ketterlin et al. (1995), which used the entity relationship model in the clustering process. One of the limitations discussed in their paper is applying their clustering system to large data sets. We intend to use CLIMIS to scale the clustering system of Ketterlin et al. (1995).

9.3.3 Extending CLIMIS to Numeric and Mixed Data

CLIMIS is based on Cobweb, which already has been extended to numeric data (Gennari, 1991) and mixed data (McKusick & Thompson, 1990). Numeric data algorithms are easier to scale and, therefore, we believe that the implementation of CLIMIS as a numeric data algorithm will be more scalable than the categorical data version of CLIMIS.

The mixed data conceptual clustering algorithm by McKusick & Thompson (1990) computes category utility for numeric data, category utility for categori-

cal data and then adds the two numbers to obtain mixed data category utility: $CU_{mixed} = CU_{categorical} + CU_{numeric}$. For its categorical data aspect, the algorithm uses the same cluster representation as Cobweb and, therefore, suffers the same complexity as Cobweb. We believe that using our *cluster summarisation* can scale the algorithm proposed by McKusick & Thompson (1990) substantially. We intend to extend CLIMIS to numeric and mixed data.

Appendix A

Experiments with ROCK and the Congressional Votes Data Set

The following table shows clustering results on the congressional votes data set with different values of θ [1].

[1] See chapter 2, section 2.6.2.

Table A.1: θ variation, 2 Clusters

$\theta = \mathbf{0.0}$		
Cluster No.	No. of Democrats	No. of Republicans
1	140	71
2	127	97
$\theta = \mathbf{0.1}$		
Cluster No.	No. of Democrats	No. of Republicans
1	4	0
2	263	168
$\theta = \mathbf{0.2}$		
Cluster No.	No. of Democrats	No. of Republicans
1	0	2
2	267	166
$\theta = \mathbf{0.3}$		
Cluster No.	No. of Democrats	No. of Republicans
1	224	9
2	43	159
$\theta = \mathbf{0.4}$		
Cluster No.	No. of Democrats	No. of Republicans
1	0	1
2	267	167
$\theta = \mathbf{0.5}$		
Cluster No.	No. of Democrats	No. of Republicans
1	1	0
2	266	168
$\theta = \mathbf{0.6}$		
Cluster No.	No. of Democrats	No. of Republicans
1	19	9
2	248	159
$\theta = \mathbf{0.7}$		
Cluster No.	No. of Democrats	No. of Republicans
1	255	36
2	12	132
$\theta = \mathbf{0.8}$		
Cluster No.	No. of Democrats	No. of Republicans
1	267	158
2	0	10
$\theta = \mathbf{0.9}$		
Cluster No.	No. of Democrats	No. of Republicans
1	267	158
2	0	10
$\theta = \mathbf{1.0}$		
Cluster No.	No. of Democrats	No. of Republicans
1	267	158
2	0	10

Appendix B

Experiments with ROCK and the Mushroom Data Set

The following table shows clustering results on the mushroom data set with different values of θ [1].

[1] See chapter 2, section 2.6.2.

Table B.1: θ variation, 3 Clusters

$\theta = \mathbf{0.5}$		
Cluster No.	Edible	Poisonous
1	41	152
2	13	23
3	4154	3741
$\theta = \mathbf{0.6}$		
Cluster No.	Edible	Poisonous
1	3743	2529
2	13	23
3	452	1364
$\theta = \mathbf{0.7}$		
Cluster No.	No. of Edible	Poisonous
1	3765	2595
2	13	23
3	430	1298
$\theta = \mathbf{0.8}$		
Cluster No.	No. of Edible	No. of Poisonous
1	3765	2595
2	13	23
3	430	1298
$\theta = \mathbf{0.9}$		
Cluster No.	No. of Edible	No. of Poisonous
1	4208	2152
2	0	36
3	0	1728

Appendix C

Experiments with LIMBO and Different ϕ Values

The following table shows results after running LIMBO with different values of ϕ [1].

[1] See chapter 2, section 2.6.3.

Table C.1: ϕ variation - 100K tuples - Requested Clusters = 10

ϕ	No. of Nodes Phase1 Produced	Returned requested clusters?
0.0	core dumped	No
0.1	core dumped	No
0.2	core dumped	No
0.3	core dumped	No
0.4	core dumped	No
0.5	core dumped	No
0.6	1	No
0.7	81	Yes
0.8	1	No
0.9	1	No
1.0	1	No
1.1	1	No
1.2	1	No
1.3	1	No
1.4	1	No
1.5	1	No

Appendix D

Data Set Example Produced with the Datgen Data Generator

The following table shows a simple data set with 20 tuples, 5 attributes and 2 clusters produced with the datgen data generator.

A	B	C	D	E	Class
a	d	c	c	b	c2
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
a	d	c	c	b	c2
c	a	b	d	a	c1
c	a	b	d	a	c1
c	a	b	d	a	c1
a	d	c	c	b	c2
a	d	c	c	b	c2
a	d	c	c	b	c2
c	a	b	d	a	c1
a	d	c	c	b	c2
a	d	c	c	b	c2
c	a	b	d	a	c1
a	d	c	c	b	c2

Table D.1: Simple Synthetic Data Set Example

References

Abdu, E. & Salane, D. (2009), A spectral-based clustering algorithm for categorical data using data summaries, *in* C. Ding & T. Li, eds, 'DMMT 2009, Proceedings of the 2nd Workshop on Data Mining using Matrices and Tensors', Paris, France, ACM, pp. 2–1–2–8.

Agrawal, R., Gehrke, J., Gunopulos, D. & Raghavan, P. (1998), Automatic subspace clustering of high dimensional data for data mining applications, *in* L. M. Haas & A. Tiwary, eds, 'SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data', Seattle, Washington, USA, ACM Press, pp. 94–105.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H. & Verkamo, A. I. (1996), Fast discovery of association rules, *in* M. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy, eds, 'Advances in Knowledge Discovery and Data Mining', American Association for Artificial Intelligence, pp. 307–328.

Al-Harbi, S. H. & Rayward-Smith, V. J. (2006), 'Adapting k-means for supervised clustering', *Applied Intelligence* **24**(3), 219–226.

Al-Zoubi, M. B. (2009), 'An effective clustering-based approach for outlier detection', *European Journal of Scientific Research* **28**(2), 310–316.

Andritsos, P. (2004), Scalable Clustering of Categorical Data and Applications, PhD thesis, University of Torondo.

Andritsos, P., Tsaparas, P., Miller, R. J. & Sevcik, K. C. (2004), Limbo: Scalable clustering of categorical data, *in* E. Bertino, S. Christodoulakis, D. Plexousakis, V. Christophides, M. Koubarakis, K. Böhm & E. Ferrari, eds, 'Advances in Database Technology - EDBT 2004: 9th International Conference on Extending Database Technology', Heraklion, Crete, Greece, Springer, pp. 123–146.

Anton, J., Jacobs, L., Liu, X., Parker, J., Zeng, Z. & Zhong, T. (2002), Web caching for database applications with oracle web cache, *in* M. J. Franklin, B. Moon & A. Ailamaki, eds, 'Proceedings of the 2002 ACM SIGMOD International Conference on Management of data', Madison, Wisconsin, USA, ACM Press, pp. 594–599.

Barbará, D., Couto, J. & Li, Y. (2002), Coolcat: An entropy-based algorithm for categorical clustering, *in* C. Nicholas, D. Grossman, K. Kalpakis, S. Qureshi, H. van Dissel & L. Seligman, eds, 'Proceedings of the Eleventh International Conference on Information and Knowledge Management', Washington, DC, New York, NY: ACM, pp. 582–589.

Basu, S., Bilenko, M., Banerjee, A. & Mooney, R. J. (2006), Probabilistic semi-supervised clustering with constraints, *in* O. Chapelle, B. Schoelkopf & A. Zien, eds, 'Semi-Supervised Learning', MIT Press, Cambridge, MA, pp. 73–102.

Béjar, J., Cortés, U., Sanguesa, R. & Poch, M. (1997), Experiments with domain knowledge in knowledge discovery, *in* H. F. Arner, ed., 'Proceedings of the

First International Conference and Exhibition on the Practical Application of Knowledge Discovery and Data Mining', London, UK, pp. 65–78.

Berkhin, P. (2002), 'Survey of clustering data mining techniques', Accrue Software, Available from: http://citeseer.ist.psu.edu/berkhin02survey.html.

Berry, M. J. A. & Linoff, G. S. (2004), *Data Mining Techniques, For Marketing, Sales and Customer Relationship Management*, 2nd edn, John Wiley & Sons.

Biswas, G., Weinberg, J. B. & Fisher, D. H. (1998), 'Iterate: A conceptual clustering algorithm for data mining', *IEEE Transactions on Systems, Man, Cybernetics - Part C: Applications and Reviews* **28**(2), 219–229.

Biswas, G., Weinberg, J., Yang, Q. & Koller, G. (1991), Conceptual clustering and exploratory data analysis, *in* L. A. Birnbaum & G. C. Collins, eds, 'Proceedings of the Eighth International Workshop on Machine Learning', San Francisco, CA: Morgan Kaufmann, pp. 591–595.

Boley, D. (2001), A scalable hierarchical algorithm for unsupervised clustering, *in* R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar & R. R. Namburu, eds, 'Data Mining for Scientific and Engineering Applications', Kluwer Academic Publishers, pp. 383–400.

Bouchachia, A. & Pedrycz, W. (2006), 'Data clustering with partial supervision', *Data Mining and Knowledge Discovery* **12**(1), 47–78.

Brackman, R. J. & Anand, T. (1996), The process of knowledge discovery in databases, *in* M. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy,

eds, 'Advances in Knowledge Discovery and Data Mining', AAAI/The MIT press, pp. 37–57.

Bradley, P., Fayyad, U. & Reina, C. (1998), Scaling clustering algorithms to large databases, *in* R. Agrawal & P. Stolorz, eds, 'Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining', New York City, AAAI Press, pp. 27–31.

Ceri, S., Cochrane, R. J. & Widom, J. (2000), Practical applications of triggers and constraints: Successes and lingering issues, *in* A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter & K. Whang, eds, 'Proceedings of the 26th International Conference on Very Large Databases', Cairo, Egypt, Morgan Kaufmann, pp. 254–262.

Chan, P. K. & Stolfo, S. J. (1993), Experiments in multistrategy learning by meta-learning, *in* B. K. Bhargava, T. W. Finin & Y. Yesha, eds, 'Proceedings of the Second International Conference on Information and Knowledge Management', Washington, DC, ACM, pp. 314–323.

Chapman, P., Clinton, J., Khabaza, T., Reinartz, T. & Wirth, R. (1999), 'The crisp-dm process model', Partners of the CRISP-DM consortium: NCR Systems Engineering Copenhagen (Denmark), DaimlerChrysler AG (Germany), Integral Solutions Ltd. (England) and OHRA Verzekeringen en Bank Groep B.V (The Netherlands), Available from: http://www.spss.it/download/pub-paper.pdf.

Chaudhuri, S. (1998), 'Data mining and database systems: Where is the intersection?', *Data Engineering Bulletin* **21**(1), 4–8.

Chaudhuri, S., Fayyad, U. & Bernhardt, J. (1999), Scalable classification over sql databases, *in* 'Proceedings of the 15th International Conference on Data Engineering', Sydney, Australia, Los Alamitos, California: IEEE Computer Society Press, pp. 470–479.

Chen, K. & Liu, L. (2005), The 'best k' for entropy-based categorical clustering, *in* J. Frew, ed., 'Proceedings of the 17th International Conference on Scientific and Statistical Database Management', Santa Barbara, CA, pp. 253–262.

Chen, Q., Wu, X. & Zhu, X. (2004), Oidm: Online interactive data mining, *in* R. Orchard, C. Yang & M. Ali, eds, 'Proceedings of the 17th International Conference on Innovations in Applied Artificial Intelligence', Ottawa, Canada, Springer, pp. 66–76.

Clear, J., Dunn, D., Harvey, B., Heytens, M. L., Lohman, P., Mehta, A., Melton, M., Rohrberg, L., Savasere, A., Wehrmeister, R. M. & Xu, M. (1999), Nonstop sql/mx primitives for knowledge discovery, *in* 'Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', San Diego, CA, USA, ACM, pp. 425–429.

Clementine (2005), *Clementine 10.0 User's Guide*, Integral Solutions Limited, Chicago, IL.

Connolly, T., Begg, C. & Strachan, A. (1996), *Database Systems - A Practical Approach to Design Implementation and Management*, Addison-Wesley Longman.

Cover, T. M. & Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley and Sons.

Cristofor, D. & Simovici, D. A. (2002), An information-theoretical approach to clustering categorical databases using genetic algorithms, *in* 'Second SIAM ICDM, Workshop on Clustering High Dimensional Data', Arlington, VA, pp. 37–46.

Date, C. J. (1995), *An Introduction to Database Systems*, 6th edn, Addison-Wesley.

Dehuri, S., Ghosh, A. & Mall, R. (2006), 'Genetic k-medoid clustering', *The ICFAI Journal of Information Technology* **Vol. II**(1), 17–28, March.

Deogun, J. S., Raghavan, V. V., Sarkar, A. & Sever, H. (1997), Data mining: Research trends, challenges, and applications, *in* T. Y. Lin & N. Cercone, eds, 'Rough Sets and Data Mining: Analysis of Imprecise Data', Kluwer Academic Publishers, pp. 9–45.

Deschler, K. W. & Rundensteiner, E. A. (2001), B+ retake: Sustaining high volume inserts into large data pages, *in* 'Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP', Atlanta, Georgia, USA, New York, NY, ACM, pp. 56–63.

Dougherty, J., Kohavi, R. & Sahami, M. (1995), Supervised and unsupervised discretization of continuous features, *in* A. Prieditis & S. J. Russell, eds, 'Proceedings of the Twelfth International Conference on Machine Learning', Tahoe City, California, USA, Morgan Kaufmann, pp. 194–202.

Dragut, A. & Nichitiu, C. (2004), 'A monotonic on-line linear algorithm for hierarchical agglomerative classification', *Information Technology and Management* **5**(1-2), 111–141, January-April.

Duda, R. & Hart, P. (1973), *Pattern Classification and Scene Analysis*, New York, NY: John Wiley & Sons.

Dunham, M. H. (2003), *Data Mining, Introductory and Advanced Topics*, Prentice Hall.

Dunkel, B. & Soparkar, N. (1999), Data organization and access for efficient data mining, *in* 'Proceedings of the 15th International Conference on Data Engineering', Sydney, Australia, IEEE Computer Society Press, pp. 522–529.

Dunkel, B., Soparkar, N., Szaro, J. & Uthurusamy, R. (1997), 'Systems for kdd: From concepts to practice', *Future Generation Computer Systems* **13**(2-3), 231–242.

Elmasri, R. & Navathe, S. B. (2000), *Fundamentals of Database Systems*, 3rd edn, Addison-Wesley.

Ester, M., Kriegel, H. P., Sander, J. & Xu, X. (1996), A density-based algorithm for discovering clusters in large spatial databases with noise, *in* E. Simoudis, J. Han & U. M. Fayyad, eds, 'Proceedings of the Second International Conference on Knowledge Discovery and Data Mining', Portland, Oregon, AAAI Press, pp. 226–213.

Everitt, B. S. (1993), *Cluster Analysis*, 2nd edn, New York: Halsted Press.

Fasulo, D. (1999), 'An analysis of recent work on clustering algorithms', University of Washington, Available from: http://citeseer.nj.nec.com/fasulo99analysi.html.

Fayyad, U., Piatesky-Shapiro, G. & Smyth, P. (1996), Knowledge discovery and data mining, towards a unifying framework, *in* E. Simoudis, J. Han & U. M. Fayyad, eds, 'Proceedings of the Second International Conference on Knowledge Discovery and Data Mining', Portland, Oregon, AAAI Press, pp. 82–88.

Fisher, D. H. (1987), 'Knowledge acquisition via incremental conceptual clustering', *Machine Learning* **2**(2), 139–172.

Fisher, D. H. (1989), Noise tolerant conceptual clustering, *in* 'Proceedings of the Eleventh International Joint Conference on Artificial Intelligence', Vol. 1, Detroit, MI: Morgan Kaufmann, pp. 825–830.

Fisher, D. H. & Langley, P. (1986), Conceptual clustering and its relation to numerical taxonomy, *in* W. A. Gale, ed., 'Artificial Intelligence and Statistics', Boston, MA: Addison-Wesley, pp. 77–116.

Fisher, D. H., Xu, L. & Zard, N. (1992), Ordering effects in clustering, *in* D. Sleeman & P. Edwards, eds, 'Proceedings of the Ninth International Workshop on Machine Learning', Aberdeen, Scotland, UK, Morgan Kaufmann, pp. 162–168.

Fraley, C. & Raftery, A. E. (1998), 'How many clusters? which clustering method? answers via model-based cluster analysis', *The Computer Journal* **41**(8), 578–588.

Frawley, W. J., Piatesky-Shapiro, G. & Matheus, C. J. (1992), 'Knowledge discovery in databases: An overview', *AI Magazine* **13**(3), 57–70, Fall.

Freitas, A. A. & Lavington, S. H. (1996), Using sql primitives and parallel db servers to speed up knowledge discovery in large relational databases, *in*

R. Trappl, ed., 'Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research', Vienna, Austria, pp. 955–960.

Gan, Q. & Suel, T. (2009), *in* 'Proceedings of the Eighteenth International Conference on World Wide Web', Madrid, Spain, ACM, pp. 431–440.

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

Garner, S. (1995), Weka: The waikato environment for knowledge analysis, *in* S. Reeves & S. Cranefield, eds, 'Proceedings of the Second New Zealand Computer Science Research Students' Conference', Hamilton, New Zealand, pp. 57–64.

Geist, I. & Sattler, K. (2002), Towards data mining operators in database systems: Algebra and implementation, *in* '2nd International Workshop on Databases, Documents and Information Fusion (DBFusion 2002)', Vol. 124, Karlsruhe, Germany, Available from: http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-124/.

Gennari, J. H. (1989), Focused concept formation, *in* 'Proceedings of the Sixth International Workshop on Machine Learning', Ithaca, NY: Morgan Kaufmann, pp. 379–382.

Gennari, J. H. (1991), Concept formation and attention, *in* 'Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society', Irvine, CA, Chicago, IL: Lawrence Erlbaum, pp. 724–728.

Gennari, J. H., Langley, P. & Fisher, D. (1989), Models of incremental concept formation, *in* J. Corbonell, ed., 'Machine Learning: Paradigms and Methods', Vol. 40, North-Holland: Elsevier, pp. 11–61.

Gibson, D., Kleinberg, J. & Raghavan, P. (2000), 'Clustering categorical data: An approach based on dynamical systems', *The VLDB Journal* **8**(3-4), 222–236.

Gluck, M. A. & Corter, J. E. (1985), Information, uncertainty and the utility of categories, *in* 'Proceedings of the Seventh Annual Conference of the Cognitive Science Society', Irvine, CA: Lawrence Erlbaum, pp. 283–287.

Goil, S., Nagesh, H. & Choudhary, A. (1999), Mafia: Efficient and scalable subspace clustering for very large data sets, Center for Parallel and Distributed Computing, CPDC-TR-9906-010, Available from: http://citeceerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.8684.

Gowda, K. C. & Diday, E. (1992), 'Symbolic clustering using a new similarity measure', *IEEE Transactions on Systems, Man and Cybernetics* **22**(2), 368–378, March/April.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. & Pirahesh, H. (1997), 'Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals', *Data Mining and Knowledge Discovery Journal* **1**(1), 29–53.

Guha, S., Rastogi, R. & Shim, K. (2000), 'Rock: A robust clustering algorithm for categorical attributes', *Information Systems* **25**(5), 345–366.

Han, J. (1998), 'Towards on-line analytical mining in large databases', *ACM SIGMOD Record* **27**(1), 97–107.

Han, J. & Kamber, M. (2000), *Data Mining Concepts and Techniques*, Morgan Kaufmann.

Harper, G. & Pickett, S. D. (2006), 'Methods for mining hts data', *Drug Discovery Today* **11**(15), 694–699, 16 August.

Hartigan, J. A. (1975), *Clustering Algorithms*, New York: Wiley.

Hastie, T., Tibshirani, R. & Friedman, J. (2001), *The Elements of Statistical Learning, Data mining, Inference and Prediction*, New York: Springer-Verlag.

He, Z., Deng, S. & Xu, X. (2006), Approximation algorithms for k-modes clustering, *in* D. S. Huang, K. Li & G. W. Irwin, eds, 'Computational Intelligence', Vol. 4114, Berlin/Heidelberg: Springer, pp. 296–302.

He, Z., Xu, X. & Deng, S. (2005), 'Scalable algorithms for clustering large datasets with mixed type attributes', *International Journal of Intelligent Systems* **20**(10), 1077–1089.

Hinneburg, A. & Keim, D. (1999), Clustering methods for large databases: From the past to the future, *in* A. Delis, C. Faloutsos & S. Ghandeharizadeh, eds, 'Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data', Philadelphia, PA, USA, New York, NY: ACM Press, p. 509.

Hipp, J., Guntzer, U. & Grimmer, U. (2001), Integrating association rule mining algorithms with relational database systems, *in* 'Proceedings of the 3rd Inter-

national Conference on Enterprise Information Systems (ICEIS 2001)', Vol. 1, Setubal, Portugal, pp. 130–137.

Holsheimer, M., Kersten, M., Mannila, H. & Toivonen, H. (1995), A perspective on databases and data mining, *in* U. M. Fayyad & R. Uthurusamy, eds, 'Proceedings of the First International Conference on Knowledge Discovery and Data Mining', Montreal, Canada, AAAI Press, pp. 150–155.

Huang, Z. (1997), A fast clustering algorithm to cluster very large categorical data sets in data mining, *in* 'Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)', Tucson, Arizona, Available from: http://www.cs.sfu.ca/news/conferences/dmkd97.html.

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

Hurst, N., Marriott, K. & Moulder, P. (2003), Cobweb: a constraint-based web browser, *in* M. J. Oudshoorn, ed., 'Twenty-sixth Australian computer science conference (ACSC 2003)', Vol. 16, Adelaide, South Australia, Australian Computer Society, pp. 247–254.

Iba, W. & Langley, P. (2001), Unsupervised learning of probabilistic concept hierarchies, *in* G. Paliouras, V. Karkaletsis & C. D. Spyropoulos, eds, 'Machine Learning and Its Applications', Berlin: Springer, pp. 39–70.

Jain, A. K. & Dubes, R. C. (1988), *Algorithms for Clustering Data*, New Jersey: Prentice-Hall.

Jaragh, M. & Hasswa, A. (2005), 'Implementation, analysis and performance evaluation of the irp replacement policy', *Microelectronics Reliability* **45**(7-8), 1264–1269, July-August.

Jonyer, I., Holder, L. & Cook, D. J. (2002), 'Graph-based hierarchical conceptual clustering in structural databases', *the journal of Machine Learning Research* **2**, 19–43, March.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R. & Wu, A. Y. (2002), 'An efficient k-means clustering algorithm: Analysis and implementation', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(7), 881–892, July.

Katayama, N. & Satoh, S. (1997), The sr-tree: an index structure for high-dimensional nearest neighbor queries, *in* J. Peckham, ed., 'SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data', Tucson, Arizona, USA, ACM Press, pp. 369–380.

Kaufman, L. & Rousseuw, P. J. (1990), *Finding Groups in Data - An Introduction to Cluster Analysis*, New York: Willey.

Kepner, J. & Kim, R. (2003), 'Cluster detection in databases: The adaptive matched filter algorithm and implementation', *Data Mining and Knowledge Discovery Journal* **7**(1), 57–79.

Ketterlin, A., Gancarski, P. & Korczak, J. J. (1995), Conceptual clustering in structured databases: A practical approach, *in* U. M. Fayyad & R. Uthurusamy, eds, 'Proceedings of the First International Conference on Knowledge Discovery and Data Mining', Montreal, Canada, AAAI Press, pp. 180–185.

Kilander, F. (1993), Cobbit - a control procedure for cobweb in the presence of concept drift, *in* P. Brazdil, ed., 'Proceedings of the European Conference on Machine Learning (ECML)', Vienna, Austria, Springer, pp. 244–261.

Knuth, D. (1973), *The Art of Computer Programming, Volume 3: Sorting and Searching*, Addison-Wesley.

Koch, G. & Loney, K. (1995), *Oracle - The Complete Reference*, McGraw-Hill.

Kumar, N., Kummamuru, K. & Paranjpe, D. (2005), Semisupervised clustering with metric learning using relative comparisons, *in* 'Proceedings of the 5th IEEE International Conference on Data Mining', Houston, Texas, USA, IEEE Computer Society, pp. 693–696.

Langley, P. (1995), Order effects in incremental learning, *in* P. Reimann & H. Spada, eds, 'Learning in Humans and Machines: Towards an Interdisciplinary Learning Science', Pergamon, pp. 154–167.

Lebowitz, M. (1987), 'Experiments with incremental concept formation: Unimem', *Machine Learning* **2**(2), 103–138.

Lee, C. (1993), Sampling issues in generating rules from databases, *in* 'Proceedings of the Fifth International Conference on Tools with Artificial Intelligence', Boston, Massachusetts, IEEE Computer Society, pp. 435–439.

Lee, D., Choi, J., Kim, J. H., Noh, S. H., Min, S. L., Cho, Y. & Kim, C. S. (2001), 'Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies', *IEEE Transactions on Computers* **50**(12), 1352–1361.

Lepinioti, K. & McKearney, S. (2007), Integrating cobweb with a relational database, *in* S. Ao, O. Castillo, C. Douglas, D. D. Feng & J. Lee, eds, 'Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS 2007)', Hong Kong, China, Newswood Limited, pp. 868–873.

Lepinioti, K. & McKearney, S. (2008), Cobweb/idx: Mapping cobweb to sql, *in* O. Castillo, L. Xu & S. Ao, eds, 'Trends in Intelligent Systems and Computer Engineering', Vol. 6, Springer, pp. 363–373.

Li, C. & Biswas, G. (2002), 'Unsupervised learning with numeric-and-nominal mixed data', *IEEE Transactions on Knowledge and Data Engineering* **14**(4), 673–690.

Li, M., Holmes, G. & Pfahringer, B. (2005), Clustering large datasets using cobweb and k-means in tandem, *in* G. I. Webb & X. Yu, eds, 'Proceedings of the 17th Australian Joint Conference on Artificial Intelligence', Cairns, Australia, Springer, Berlin, pp. 368–379.

Li, T. (2005), A general model for clustering binary data, *in* R. Grossman, R. J. Bayardo & K. P. Bennett, eds, 'Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', Chicago, Illinois, USA, ACM, pp. 188–197.

Liu, H., Lu, H. & Chen, J. (2002), 'A fast scalable classifier tightly integrated with rdbms', *Journal of Computer Science and Technology* **17**(2), 152–159.

Liu, Y., Liao, W., Choudhary, A. & Li, J. (2008), Parallel data mining algorithms for association rules and clustering, *in* S. Rajasekaran & J. Reif, eds, 'Handbook of Parallel Computing', Chapman and Hall/CRC, pp. 32–1–32–20.

Lu, H. (2001), Seamless integration of data mining with dbms and applications, *in* D. W. Cheung, G. J. Williams & Q. Li, eds, 'Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2001)', Hong Kong, China, Springer, p. 3.

Lu, H., Yuan, S. & Lu, S. Y. (1996), On preprocessing data for effective classification, *in* 'Workshop on Research Issues on Data Mining and Knowledge Discovery', Montreal, Canada, Available from: http://fas.sfu.ca/cs/conf/dmkd96.html.

MacQueen, J. B. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability (5th)', Vol. 1, Berkeley Calif.: University of California Press, pp. 281–297.

Martin, J. D. (1994), Dp1: Supervised and unsupervised clustering, *in* F. Bergadano & L. D. Raedt, eds, 'Proceedings of the European Conference on Machine Learning (ECML-94)', Catania, Italy, Springer, pp. 395–398.

McCallum, A., Nigam, K. & Ungar, L. H. (2000), Efficient clustering of high-dimensional data sets with application to reference matching, *in* 'Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', Boston, MA, USA, ACM, pp. 169–178.

McKusick, K. & Thompson, K. (1990), Cobweb/3: A portable implementation, NASA Ames Research Center, Artificial Intelligence Research Branch, FIA-90-6-18-2, Available from: http://citeceerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.4676.

Mehta, M., Agrawal, R. & Rissanen, J. (1996), Sliq: A fast scalable classifier for data mining, *in* P. M. G. Apers, M. Bouzeghoup & G. Gardarin, eds, 'Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology', Avignon, France, Springer, pp. 18–32.

Michalski, R. S. (1980), 'Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts', *International Journal of Policy Analysis and Information Systems* **4**(3), 219–244.

Michalski, R. S. & Kaufman, K. A. (1998), Data mining and knowledge discovery: A review of issues and a multistrategy approach, *in* R. S. Michalski, I. Bratko & M. Kubat, eds, 'Machine Learning and Data Mining Methods and Applications', John Wiley, England, pp. 71–112.

Michalski, R. S. & Stepp, R. E. (1983), Learning from observation: conceptual clustering, *in* R. S. Michalski, J. G. Carbonell & T. M. Mitchell, eds, 'Machine Learning: An Artificial Intelligence Approach', San Mateo, CA: Morgan Kaufmann, pp. 331–364.

Milligan, G. W. (1996), Clustering validation: Results and implications for applied analyses, *in* P. Arabie, L. J. Hubert & G. D. Soete, eds, 'Clustering and Classification', River Edge, NJ: World Scientific Publ., pp. 341–371.

Mirkin, B. (2005), *Clustering for Data Mining, A Data Recovery Approach*, Chapman & Hall/CRC.

Netz, A., Chaudhuri, S., Bernhardt, J. & Fayyad, U. (2000), Integration of data mining and relational databases, *in* A. E. Abbadi, M. L. Brodie,

S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter & K. Whang, eds, 'Proceedings of the 26th International Conference on Very Large Databases', Cairo, Egypt, Morgan Kaufmann, pp. 719–722.

Ng, R. T. & Han, J. (1994), Efficient and effective clustering methods for spatial data mining, *in* J. B. Bocca, M. Jarke & C. Zaniolo, eds, 'Proceedings of the 1994 International Conference on Very Large Databases (VLDB'94)', Santiago, Chile, Morgan Kaufmann, pp. 144–155.

Ng, R. T. & Han, J. (2002), 'Clarans: A method for clustering objects for spatial data mining', *IEEE Transactions on Knowledge and Data Engineering* **14**(5), 1003–1016, September/October.

Nguyen, Q. H. & Rayward-Smith, V. J. (2008), 'Internal quality measures for clustering in metric spaces', *Int. J. Business Intelligence and Data Mining* **3**(3), 4–29.

Nittel, S., Leung, K. & Braverman, A. (2004), Scaling clustering algorithms for massive data sets using data streams, *in* 'Proceedings of the 20th International Conference on Data Engineering (ICDE 2004)', Boston, MA, USA, IEEE Computer Society, p. 830.

Oates, T. (2002), Peruse: An unsupervised algorithm for finding recurring patterns in time series, *in* 'Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)', Maebashi City, Japan, IEEE Computer Society, pp. 330–337.

Ordonez, C. (2006), 'Integrating k-means clustering with a relational dbms using sql', *IEEE Transactions on Knowledge and Data Engineering* **18**(2), 188–201.

Ordonez, C. & Omiecinski, E. (2004), 'Efficient disk-based k-means clustering for relational databases', *IEEE Transactions on Knowledge and Data Engineering* **16**(8), 909–921.

Paliouras, G., Papatheodorou, C., Karkaletsis, V., Spyropoulos, C. & P.Tzitziras (1999), From web usage statistics to web usage analysis, *in* 'Proceedings of the IEEE International Conference on Systems, Man and Cybernetics', Vol. 2, Tokyo, Japan, pp. 159–164.

Pelleg, D. & Moore, A. W. (2000), X-means: Extending k-means with efficient estimation of the number of clusters, *in* P. Langley, ed., 'Proceedings of the 17th International Conference on Machine Learning', Standord, CA, USA, Morgan Kaufmann, pp. 727–734.

Perkowitz, M. & Etzioni, O. (2000), 'Towards adaptive web sites: Conceptual framework and case study', *Artificial Intelligence* **118**(1), 245–275.

Pyle, D. (1999), *Data Preparation for Data Mining*, Morgan Kaufmann.

Rajamani, K. & Cox, A. (1999), Efficient mining for association rules with relational database systems, *in* 'Proceedings of the International Database Engineering and Applications Symposium', Montreal, Canada, IEEE Computer Society, pp. 148–155.

Ramakrishnan, R. (2003), *Database Management Systems*, 3rd edn, McGraw-Hill Companies, Inc.

Rayward-Smith, V. J. (2007), 'Statistics to measure correlation for data mining applications', *Computational Statistics & Data Analysis* **51**, 3968–3982.

Ribeiro, J. S., Kaufman, K. A. & Kerschberg, L. (1995), Knowledge discovery from multiple databases, *in* U. M. Fayyad & R. Uthurusamy, eds, 'Proceedings of the First International Conference on Knowledge Discovery and Data Mining', Montreal, Canada, AAAI Press, pp. 240–245.

Roure, J. & Talavera, L. (1998), Robust incremental clustering with bad orderings: A new strategy, *in* H. Coelho, ed., 'Proceedings of the 6th Ibero-american Conference on Artificial Intelligence - IBERAMIA 98', Vol. 1484, Lisbon, Portugal, Springer, pp. 136–147.

Sarawagi, S., Thomas, S. & Agrawal, R. (1998), Integrating mining with relational database systems: Alternatives and implications, *in* L. M. Haas & A. Tiwary, eds, 'Proceedings ACM SIGMOD International Conference on Management of Data', Seattle, Washington, USA, ACM Press, pp. 343–354.

Sarawagi, S., Thomas, S. & Agrawal, R. (2000), 'Integrating association rule mining with relational database systems: Alternatives and implications', *Data Mining and Knowledge Discovery* **4**(2-3), 89–125, July.

Sattler, K. & Dunemann, O. (2001), Sql database primitives for decision tree classifiers, *in* 'Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management', Atlanta, Giorgia, USA, ACM, pp. 379–386.

Sellis, T. K., Roussopoulos, N. & Faloutsos, C. (1987), The r+-tree: A dynamic index for multi-dimensional objects, *in* W. K. P. M. Stocker & P. Hammersley, eds, 'Proceedings of 13th International Conference on Very Large Data Bases (VLDB'87)', Brighton, England, Morgan Kaufmann, pp. 507–518.

Selvakumar, S., Kumar & Venkatasubramani, V. (2004), 'Delay sensitive least frequently used algorithm for replacement in web caches', *Computer Communications* **27**(3), 322–326, February.

Serazi, M., Perera, A., Ding, Q., Malakhov, V., Rahal, I., Pan, F., Ren, D., Wu, W. & Perrizo, W. (2004), Datamine, *in* G. Weikum, A. C. König & S. Deßloch, eds, 'Proceedings of the ACM SIGMOD International Conference on Management of Data', Paris, France, ACM, pp. 923–924.

Shafer, J., Agrawal, R. & Mehta, M. (1996), Sprint: A scalable parallel classifier for data mining, *in* T. M. Vijayaraman, A. P. Buchmann, C. Mohan & N. L. Sarda, eds, 'Proceedings of the 22nd International Conference on Very Large Databases', Mumbai (Bombay), India, Morgan Kaufmann, pp. 544–555.

Shannon, C. E. & Weaver, W. (1949), *The Mathematical Theory of Communication*, Univ. of Illinois Press.

Shatdal, A., Kant, C. & Naughton, J. (1994), Cache conscious algorithms for relational query processing, *in* J. B. Bocca, M. Jarke & C. Zaniolo, eds, 'Proceedings of the 20th International Conference on Very Large Data Bases, VLDB'94', Santiago, Chile, Morgan Kaufmann, pp. 520–521.

Shearer, C. (2000), 'The crisp-dm model: the new blueprint for data mining', *Journal of Data Warehousing* **5**(4), 13–22, fall.

Sheikholeslami, G., Chatterjee, S. & Zhang, A. (1998), Wavecluster: A multi-resolution clustering approach for very large spatial databases, *in* A. Gupta, O. Shmueli & J. Widom, eds, 'Proceedings of the 24th International Conference

on Very Large Databases', New York, NY, USA, Morgan Kaufmann, pp. 428–439.

Sison, R. & Shimura, M. (1996), Incremental clustering of relational descriptions, Department of Computer Science, Tokyo Institute of Technology, TR96-0011, Available from: http://citeceerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9906.

Skillicorn, D. B. (1999), 'Strategies for parallel data mining', *IEEE Concurrency* **7**(4), 26–35, October/December.

Slonim, N. & Tishby, N. (2000), Agglomerative information bottleneck, *in* S. A. Solla, T. K. Leen & K. R. Müller, eds, 'Advances in neural information processing systems 12', Cambridge, MA: MIT Press, pp. 617–623.

Sokolinsky, L. B. (2004), Lfu-k: An effective buffer management replacement algorithm, *in* Y. Lee, J. Li, K. Y. Whang & D. Lee, eds, 'Database Systems for Advanced Applications, DASFAA2004', Berlin/Heidelberg: Springer, pp. 670–681.

Song, Y., Goncalves, L. & Perona, P. (2003), 'Unsupervised learning of human motion models', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25**(7), 814–827, July.

Sousa, M., Mattoso, M. & Ebecken, N. (1998), Data mining: A database perspective, *in* 'Proceedings, International Conference on Data Mining', Rio de Janeiro, Brasil, WIT Press, pp. 413–432.

Srivastava, A., Han, E., Kumar, V. & Singh, V. (1999), 'Parallel formulations of

decision-tree classifications algorithms', *Data Mining and Knowledge Discovery* **3**(3), 237–261.

Stevens, S. (1946), 'On the theory of scales of measurement', *Science* **103**(2684), 677–680.

Stierhoff, G. C. & Davis, A. G. (1998), 'A history of the ibm systems journal', *IEEE Annals of the History of Computing* **20**(1), 29–35.

Surdeanu, M., Turmo, J. & Ageno, A. (2005), A hybrid unsupervised approach for document clustering, *in* R. Grossman, R. J. Bayardo & K. Bennett, eds, 'Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', Chicago, Illinois, USA, ACM, pp. 685–690.

Talavera, L. (2000), Feature selection and incremental learning of probabilistic concept hierarchies, *in* P. Langley, ed., 'Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)', Standord, CA, USA, Morgan Kaufmann, pp. 951–958.

Talavera, L. & Roure, J. (1998), A buffering strategy to avoid ordering effects in clustering, *in* C. Nedellec & C. Rouveirol, eds, 'Proceedings of the 10th European Conference on Machine Learning (ECML '98)', Chemnitz, Germany, Springer, pp. 316–321.

Theodorakis, M., Vlachos, A. & Kalamboukis, T. Z. (2004), Using hierarchical clustering to enhance classification accuracy, *in* '3rd Hellenic Conference on Artificial Intelligence (SETN 04)', Samos, Available from: http://www.icsd.aegean.gr/setn04/enprogram.htm.

Thompson, K. & Langley, P. (1991), Concept formation in structured domains, *in* D. H. Fisher, M. J. Pazzani & P. Langley, eds, 'Concept Formation: Knowledge and Experience in Unsupervised Learning', Morgan Kaufmann, pp. 127–161.

Venketesh, P., S.N., Sivanandam & S.Manigandan (2006), 'Enhancing qos in web caching using differentiated services', *International Journal of Computer Science and Applications* **III**(I), 78–91.

Wang, M., Iyer, B. R. & Vitter, J. S. (1998), Scalable mining for classification rules in relational databases, *in* B. Eaglestone, B. C. Desai & J. Shao, eds, 'Proceedings of the International Database Engineering and Application Symposium', Cardiff, UK, IEEE Computer Society, pp. 58–67.

Wang, W., Yang, J. & Muntz, R. R. (1997), Sting: A statistical information grid approach to spatial data mining, *in* M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos & M. A. Jeusfeld, eds, 'Proceedings of 23rd International Conference on Very Large Data bases', Athens, Greece, Morgan Kaufmann, pp. 186–195.

Weng, J., Zhang, Y. & Hwang, W. S. (2003), 'Candid covariance-free incremental principal component analysis', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25**(8), 1034–1040.

Witten, I. H. & Frank, E. (2000), *Data Mining - Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann.

Witten, I. H. & Frank, E. (2005), *Data Mining, Practical Machine Learning Tools and Techniques*, 2nd edn, Morgan Kaufmann.

Wu, S., Wu, G., Yu, Z. & Ban, H. (2005), A platform for parallel data mining on cluster system, *in* W. Zhang, Z. Chen, R. Glowinski & W. Tong, eds, 'Current Trends in High Performance Computing and Its Applications', Springer-Verlag Berlin Heidelberg, pp. 155–164.

Zaki, M., Peters, M., Assent, I. & Seidl, T. (2005), Clicks: An effective algorithm for mining subspace clusters in categorical datasets, *in* R. Grossman, R. J. Bayardo & K. P. Bennett, eds, 'Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', Chicago, Illinois, USA, ACM, pp. 736–742.

Zhang, Q. & Couloigner, I. (2005), A new efficient k-medoid algorithm for spatial clustering, *in* 'Computational Science and Its Applications - ICCSA 2005', Vol. 3482/2005, Berlin/Heidelberg: Springer, pp. 181–189.

Zhang, T. (1997), Data Clustering For Very Large Datasets plus Applications, PhD thesis, University of Wisconsin, Madison.

Zhang, T., Ramakrishnan, R. & Livny, M. (1996), Birch: An efficient data clustering method for very large databases, *in* H. V. Jagadish & I. S. Mumick, eds, 'Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data', Montreal, Quebec, Canada, ACM Press, pp. 103–114.

Zhang, Y., Wai-chee, A., Chun, F., Cai, H. & Heng, P. (2000), Clustering categorical data, *in* 'Proceedings of the 16th International Conference on Data Engineering', San Diego, CA, USA, IEEE Computer Society, p. 305.

Zloof, M. M. (1975), Query by example, *in* 'AFIPS Conference Proceedings', Vol. 44, Anaheim, CA, AFIPS Press, pp. 431–438.