R41
C.1

Lebanese American University

Incremental Dependency Aided Metapattern Generator

By

Nassim Z. Abdel Samad

Thesis
Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Beirut, Lebanon
July 1999

Lebanese American University

Incremental Dependency Aided Metapattern Generator

By

Nassim Z. Abdel Samad

Thesis
Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

[redacted]

Dr. Issam Moghrabi (Supervisor)
Assistant Professor of Computer Science
Lebanese American University

[redacted]

Dr. Ahmad Nasri (Reader)
Associate Professor in Computer Graphics Department
American University of Beirut

[redacted]

Dr. May Abboud (Reader)
Associate Professor of Math and Computer Science
Lebanese American University

Table of Contents

Acknowledgments

I am grateful to Dr. Issam Moghrabi for his supervision of this work and for his helpful suggestions. I thank Dr. Ahmad Nasri, American University of Beirut, for reading this work and for his helpful comments. I should also like to thank Dr. May Aboud for reading this work.

I am indebted to Mr. Raffic Harriri who financially helped me in getting my BS degree.

Finally, I thank my family for their wholehearted support and encouragement throughout the working on this thesis. They were always understanding and next to me helping. They were always concerned about establishing for me the write environment to work, and for this I am most grateful. I dedicate this work to my father, Zawakan, my mother, May, and my brother Rabih, with love and admiration.

Abstract

Accumulation of data in electronic format is increasing at an exponential rate. The increase of electronic data gathering such as remote sensing point-of-scale or even changing libraries to electronic format and books to CD's led to such an explosion. This valuable data will remain static and unexploited on secondary storage until retrieved and analyzed to acquire knowledge. Traditional ways of data manipulation are good at putting data into databases quickly, safely, and efficiently; but are not good at delivering meaningful analysis in return. Traditional On-Line Transaction Processing (OLTP) will only scratch the surface of huge and deep databases. This issue led to the need of having new intelligent techniques for analytic processing over these huge databases. This is where Data Mining has obvious benefits of combining artificial intelligence, logic programming, neural nets, machine learning and others with databases to acquire knowledge or what is also known as Knowledge Discovery in Databases (KDD). This work aim at giving an over view of data mining and enhancing a data mining technique called Metapattern generation. Domain knowledge is exploited in our approach and integrated with the original algorithm to generate interesting patterns that were not generated by the original algorithm. Further more, new techniques are incorporated to enable the original algorithm cope with incremental data.

Chapter1

Introduction

The capabilities of both generating and collecting data have been increasing rapidly. The wide spread of bar codes, the computerization of many business and government transactions and advances of data collection tools like remote sensing and others, which are used in various fields and applications have initiated the need to think seriously of new techniques, methods, and tools that can intelligently and effectively transform the acquired data into information and knowledge. As a result, data mining, which is supposed to be the most important step in the knowledge discovery process, has become a research area in many disciplines working in the fields of databases and artificial intelligence.

Data mining, which in some cases is referred to as Knowledge Discovery in Databases (KDD) while in some other cases is referred to as only one step in KDD, deals with pattern recognition and creating models. Regardless of this confusion, we have some definitions of data mining or KDD.

"*Data mining, or Knowledge Discovery in Databases as it is also known, is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. This encompasses a number of different technical approaches, such as clustering, data summarization, learning classification rules, finding dependency networks, analyzing changes, and detecting anomalies*"[1].

There are also many other definitions appearing in some articles and documents, carrying a similar or slightly different meaning, such as

"*Data mining is the search for relationships and global patterns that exist in large databases but are hidden among the vast amount of data, such as a relationship between patient data and their medical diagnosis. These relationships represent valuable knowledge about the database and the objects in the database and, if the database is a faithful mirror, of real world registered by the database*"[6].

Also, "*Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data*" and "*KDD is the process of extracting high-level knowledge from low-level data*"[2].

Mining describes a multidisciplinary field of research that includes machine learning, statistics, database technology, rule based systems, neural networks, and visualization. Looking at the practical use of data mining and knowledge acquisition, the discovered data can be applied to information management, query processing, decision making, process control, and many other applications like reverse engineering. Researchers in many different fields, including database systems, knowledge base systems, artificial intelligence, machine learning, knowledge acquisition, statistics, spatial databases, data visualization, have emerged as they are directly related to Data Mining. Several applications in information providing services, such as electronic libraries, on-line services, and World Wide Web have shown great interest in data mining and called for various data mining techniques to better understand user behavior to enhance the services provided, and increase business opportunities[3].

In our work, we will describe the data mining process and the various techniques and operations applied. Different kinds of knowledge representation, data formats and repositories are also described. In our work we will concentrate on a new technique called Metapattern generation. The Metapattern Generator algorithm was enhanced by integrating it with Induced and Fuzzy Functional Dependencies, Joined dependency and Foreign dependencies. This integration enhanced the generation of transitive patterns and made also the generation of non-transitive patterns possible.

Since Relational On-Line Analytic Processing (ROLAP) is mostly based on a star schema and other denormalized models, we made use of dependencies to be able to generate patterns from only one table (Fact table) or any other ROLAP model. In ROLAP, data can be easily updated or incremented, thus new techniques were developed to update patterns as well.

Chapter2

Data Mining Overview

2.1 Data Mining Life Cycle

Data mining refers to extracting patterns and generating models from huge data. But this is not all; many steps precede the data-mining step in knowledge discovery in databases. Transforming the contents of databases into information that can derive decision-making is a complex process that can be organized into six major steps: [2]

1. **Application understanding**: Understanding the applications' domains and their interdependencies to collect and retrieve the relevant data from large warehouses or any other kind of databases of different applications where they share some information that can be useful in the knowledge discovery process.
2. **Data selection and sampling**: Selecting the appropriate set to work with according to some criteria and deciding on the appropriate sampling strategy. A data warehouse contains a variety of diverse data not all of which will be necessary to achieve a data mining goal.
3. **Data cleaning and preprocessing**: Involved here are basic operations for cleaning the data by dealing with or removal of noise, outliers, missing fields, unnecessary data, or attributes. Also the data is reconfigured to ensure a consistent format as there is a possibility of inconsistent formats because of the data drawn from several sources (e.g., sex might be represented as F/M or 0/1).
4. **Data transformation and reduction**: the type of transformation is dependent upon the type of data mining operation or technique used. Transformation varies from conversion of one data type to another (e.g., converting nominal values into numeric ones so that they can be processed by neural networks), to derivation of new attributes. Necessary transformations, dimensionality reduction and projections are applied by reducing the number of variables using mathematical and logical operators and projecting the data to a space where the solution is easier to find (e.g. put the ratio of two attributes in one).
5. **Data Mining**: This is the most important step where intelligent techniques are used for mining the processed data by finding patterns or generating models that fit to the data. At this step, data is transformed into knowledge by applying one or several of the data mining techniques like classification, association clustering, and metapattern generation. The user can significantly aid the data mining by interfering during the generation process and correctly performing previous steps.
6. **Result Interpretation**: Evaluation and interpretation of patterns using visualization techniques and other techniques to present the knowledge discovered to the analyzer where it is easier to achieve predictive results. Achieving acceptable results may involve using:
   - Measures of interest and others to filter or prune the generated knowledge. These measures can be statistical, best fit, or heuristic measures.
   - Visualization techniques to help analysts or decision-makers reach valuable conclusions.
   - User external knowledge to screen through derived patterns and to select interesting ones.

The data mining process with the appropriate feedback steps between the various data mining operations is shown in Figure 1.

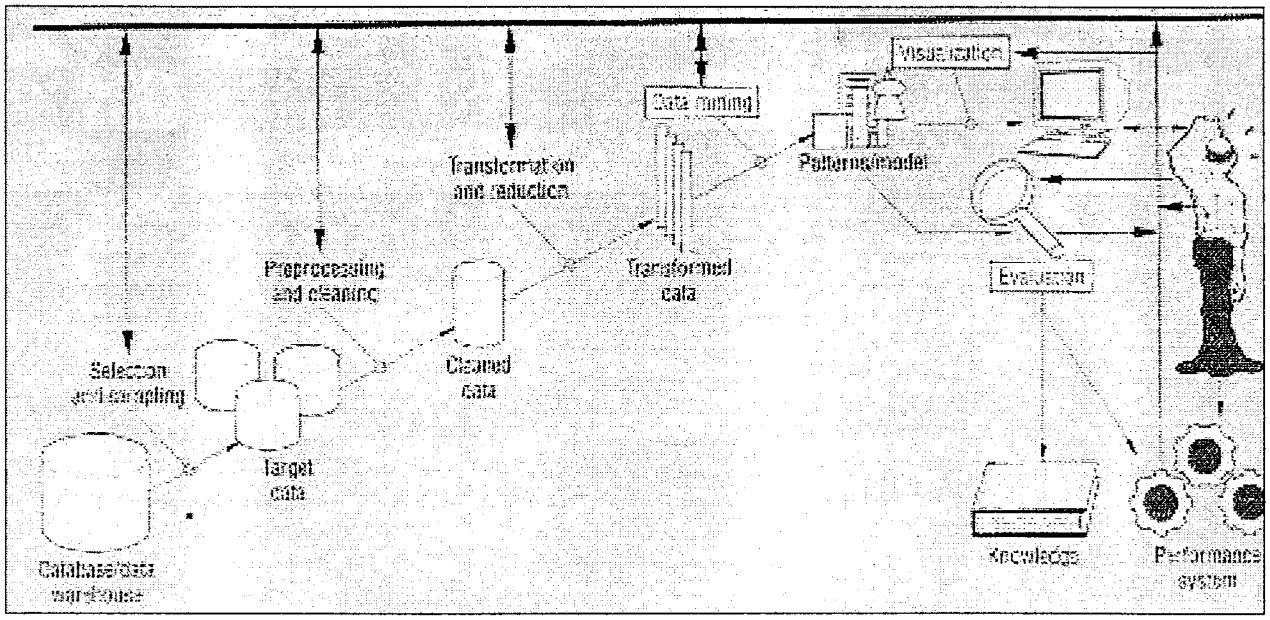


Figure 1 An overview of the KDD process [4]

2.2 *Logical Inference and learning*

Logical inference is very essential in data mining. Inference techniques are used in most of the data mining phases. The two main inference techniques are induction and deduction, which are jointly used to extract the hidden knowledge in databases.

1. Deduction is a technique to infer knowledge that is logically interrelated or information that is a logical consequence of the information in the database. Most database systems (DBMS), such as RDBMSs offer simple operations for On Line Transaction Processing (OLTP) that help in deduction of information For example, transitivity that can be applied in the form of a join between two tables. The join applied to two tables where the first concerns students and courses and the other concerns courses and instructors infer a relation between students and instructors.

2. Induction is a technique to infer Knowledge that is generalized from the database. From the example above we can infer that every student has an instructor. This general statement about the objects in the database gives us a higher level of information or knowledge. We search the database for regularities, combinations of values for certain attributes, shared by facts in the database. This regularity is a high level summary of hidden information in the database

Deduction is based on strong theoretical background and is mainly used in transaction processing rather than analysis. Work in this field is saturated where many deductive functions are part of DBMSs and other applications like deductive databases. Although deduction helps in many phases of the induction process, we will concentrate in data mining on induction rather than on deduction. Learning by induction and its various types are defined in the following sections.

2.2.1 Inductive learning

The process of simplifying the environment to a model or linking objects with similar patterns is called inductive learning. There are two types of inductive learning:

1) Supervised learning: is learning from examples where an external teacher defines classes and provides the system with examples of each class. Then the system has to find common patterns in the examples, which are the description of each class. The teacher can define a single class or multiple classes [5].

   Single Class Learning: the teacher defines a single class C for which a characteristic description has to be constructed: a description that singles out instances of C from any other example that is not an instance of C. One can distinguish between two cases: all examples are members of this class, the positive example, and the system has to find description for these examples. Alternatively, when both positive and negative examples are provided, the negative example can be seen as members of other classes.
   Multiple class learning: The teacher defines a finite number of classes $C_1, C_2, \ldots C_m$, for descriptions have to be found. The system can search for characteristic descriptions that distinguish instances in C_i from any other example. Instances of C_i form the positive examples, and all other objects not in form the negative examples. Alternatively, if each example belongs to at least one class, the system can search for discriminating descriptions that together cover all objects, and separate an instance of a class from instances of all other classes.

2) Unsupervised learning: is also known as learning from observation and discovery. Here the system is supplied with objects but no classes are defined so the system has to discover the classes itself, based on common properties of objects. Practically, the system has to find some clustering. The data mine system is supplied objects, as in supervised learning, but here no classes are defined. The system has to observe the examples, and recognize patterns by itself. Hence, this learning form is also called learning by observation and discovery [5].

2.2.2 Machine Learning

Machine learning is the automation of the inductive learning, which has been researched in the field of artificial intelligence. The machine learning system interacts with a training set, which is a coded observation and not an environment. A learning algorithm takes the data set and its accompanying information as input and returns a concept representing the results of learning as output. It also passes over previous cases and their results and learns how to reproduce these and make generalization about new cases. In other words, descriptions are constructed using iterative search strategy where the set of all constructible descriptions is searched.

Generally, a machine learning system does not use single observations of its environment, as cognitive systems do, but an entire finite set called the training set at once. This set contains examples coded in some machine-readable form. When we use a database as a training set, the learning process is called *Data Mining*. Given that the training set is finite not all concepts can be learned exactly, since some problems are undecidable. However, some of these concepts can be approximated.

2.2.3 Machine learning Vs Data Mining

Data mining is a special kind of machine learning where the environment is observed through the database. Although the frameworks for data mining and machine learning in general may seem similar, there are important differences[6].

First, the database is often designed for purposes different from data mining. In other words, it is designed for On Line Transaction Processing (OLTP) rather than On Line Analytic Processing (OLAP). That is, the representation of the real world objects in the database has been chosen to meet the needs of applications rather than the needs of Data mining. Hence, properties or attributes that would simplify the learning task are not necessarily present. Moreover, these properties can not be requested from the real world.

Second, databases are invariably contaminated by errors. Where as in machine learning the algorithm is often supplied chosen laboratory examples, in data mining the algorithm has to cope with noisy and some times contradictory data.

Third, in machine learning the training set is usually small compared to the vast amount of data collected in databases where the algorithm has to cope with the complexities that result from huge databases. Furthermore, data is not always described in structured records as it is also described in an unstructured format like documents and Web pages.

2.3 Data Mining Techniques

The techniques of data mining can be classified according to the functions they perform or according to the applications to be applied on. The most five important techniques and operations applied to data mining are the following:

2.3.1 Association

The problem of mining association rules was introduced by Agrawal in [7]. Given a set of transactions, where each transaction is a set of items, an association rule is an expression $X \Rightarrow Y$, where X and Y are sets of items. The intuitive meaning of such a rule is that transactions in the database which contain the items in X tend to also contain the items in Y is. An example might be that 98% of customers who purchase tires and auto accessories also buy some automotive services; here 98% is called the *confidence* of the rule. The *support* of the rule $X \Rightarrow Y$ is the percentage of transactions that contain both X and Y. the problem of mining association rules is to find all rules that satisfy a user-specified minimum support and minimum confidence. Applications include cross-marketing, attached mailing, catalog design, loss-leader analysis, store layout, and customer segmentation based on buying patterns.[8]

2.3.1.1 Association rules

Let $I=\{i_1, i_2,\ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. We say that a transaction T contains X, a set of some items in I, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ holds in the transaction set D with confidence c if c% of the transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support s* in the transaction set D is s% of transactions in D contain $X \cup Y$. Let Pr (X) denote the probability that all items in X are contained in a transaction. Then support $(X \Rightarrow Y) = \Pr(X \cup Y)$ and confidence $(X \Rightarrow Y) = \Pr(X \mid Y)$.[9]

Given a set transactions *D*, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user specified minimum support and minimum confidence.

The problem of mining association rules is decomposed into two subproblems:

- Find all combinations of items that have transaction support above minimum support. Call those combinations *frequent itemsets.*
- Use the frequent itemsets to generate the desired rules. The general idea is that if, say, *ABCD* and *AB* are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio r = support (*ABCD*)/support (*AB*). The rule holds only if $r \geq$ minimum confidence. Note that the rule will have minimum support because *ABCD* is frequent.

Dataset

Transaction ID	Items
1	Shoes, shirt, Jacket
2	Shoes, Jacket
3	Shoes, Jeans
4	Shirt, Sweatshirt

Frequent Itemsets

Frequent Itemset	Support
{Shoes}	75%
{Shirt}	50%
{Jacket}	50%
{Shoes, Jacket}	50%

Rules

Frequent Itemset	Support	Confidence
Shoes $\Rightarrow$ Jacket	50%	66.6%
Jacket $\Rightarrow$ Shoes	50%	100%

Figure 2 Association Rules Example[10]

Figure 2 shows an example, assuming a minimum support of 50% and minimum confidence of 50%. The first sub-problem is responsible for most of the computation time, and has been the focus of considerable work on developing fast algorithms, e.g. (Agrawal et al. 1996) (Brin et al. 1997).

2.3.1.2 Apriori Algorithm

The first pass of the algorithm simply counts item occurrences to determine the frequent 1-itemsets. A subsequent pass, pass k, consists of two phases. First, the frequent item sets L_{k-1} found in the (k-1)th pass are used to generate the candidate itemsets C_k, using the Apriori candidate generation procedure described below. Next, the database is scanned and the support of candidates in C_k is counted. For fast counting, we need to efficiently determine the candidates in C_k contained in a given transaction t.

```
$L_1$: ={frequent 1-itemsets}
$k$:= 2; //$k$ represents the pass number
while ($L_{k-1} \neq \phi$) do
begin
    $C_k$ := New candidates of size $k$ generated from $L_{k-1}$;
    For all transactions $t \in D$ do
        Increment the count of all candidates in $C_k$ that are contained in $t$;
    $L_k$ := All candidates in $C_k$ with minimum support;
    $k$: = $k$+1;
End
Answer := $\cup_k L_k$;
```

Figure 3 Apriori Algorithm.[11]

2.3.2 Classification and Supervised Induction

Data classification is the process, which finds the common properties among a set of objects in a database and classifies them into different classes according to a classification model. Predictive modeling is usually implemented by creating a classification model from a set of records (examples), called the training set usually refers to the process of supervised induction.

In the supervised learning case, this requires that the user defines one or more classes, also known as concepts, in the database [12]. The records in the training set must belong to a small set of classes that have been predefined by the analyst. The training set may either be a sample of the database or warehouse being mined, or the entire database or a data warehouse. The attributes that denote the class of tuples are called the predicted attributes. The attributes of the rule are called the predicting attributes [6].

The induced model consists of patterns, essentially generalizations over the records that are useful for distinguishing the classes. Once a model is induced it can be used to automatically predict the class of other unclassified records. Supervised induction methods can be either symbolic or neural. Symbolic methods create models that are represented either as classification rules, or as decision trees. Neural methods, such as back propagation, represent the model as architecture of nodes and weighted links. [13]

2.3.2.1 Classification rules

Data mining has to infer a model from the database by defining classes of tuples by one or more attributes. In other words, part of the attributes are predicting attributes while the remaining are the predicted attributes. The class can be defined by conditions on the attributes. After defining the classes, the rules that govern the classification should be inferred; i.e., find the description of each class. These can be represented as production rules in the form of if-then statements where the predicted attributes are the consequents and the predicting attributes lie in the antecedents

The objective of classification is first to analyze the training data and develop an accurate description or a model for each class using the features available in the data. Such class descriptions are then used to classify future test data in the database or to develop a better description called classification rules for each class in the database [3]. Rules are generated using algorithms such as IBM's RMINI, the public domain algorithm foil, etc.[13]

Classification rules are mostly in the form of "if ... then..." rules, that is, if the left-hand side (LHS) is true then the right-hand side (RHS) is true with a certain probability coefficient (confidence). The three categories are:

- Exact rule: each object of the LHS must be an element of the RHS.
- Strong rule: allows some exceptions but with a certain limit.
- Probabilistic rule: relates the conditional probability P (RHS|LHS) to the probability P (RHS)

2.3.2.2 Decision Trees

A decision-tree-based classification method is a supervised learning method that constructs decision trees from a set of examples. A decision tree is a simple knowledge representation, which classifies examples to a finite number of classes. Nodes in the tree are labeled with attribute names, the edges are labeled with possible values for these attributes, and the leaves are labeled with the different classes. An object is classified by following a path down the tree from the root to the leaf node [6]. Decision trees are generated using algorithms such as ID3 [14], and CART [15] that perform classification in two phases: Tree Building and Tree Pruning.

Tree Building: an initial decision tree is grown in this phase by repeatedly (recursively) partitioning the training data. The training set is split into two or more partitions using an attribute. The following figure gives an overview of the process.

```
MakeTree (Training Data $T$)
    Partition ($T$);

Partition (Data $S$)
    If (all points in $S$ are in the same class) then return;
    Evaluate splits for each attribute A
    Use best split found to partition $S$ into $S_1$ and $S_2$;
    Partition ($S_1$);
    Partition ($S_2$);
```

Figure 4 Tree-Building Algorithm [16]

Tree Pruning: the tree built in the first phase completely classifies the training data set. This implies that branches are created in the tree even for spurious "noise" data and statistical fluctuations. These branches can lead to errors when classifying test data. Tree pruning is aimed at removing these branches from the decision tree by selecting the sub-tree with the least estimated error rate. [16]

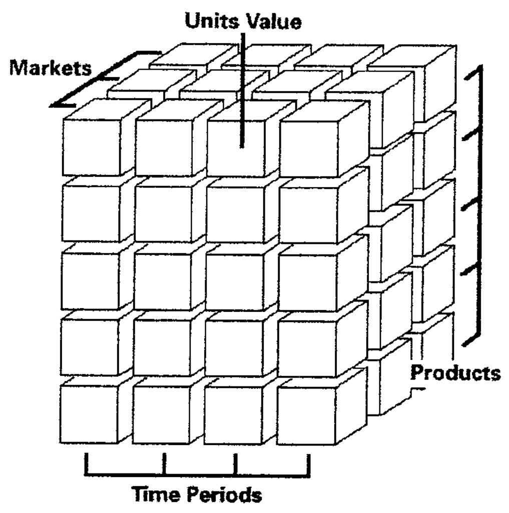


Figure 5 An Example Data cube

2.3.2.3 Rules Versus Decision Trees

Rule induction and decision trees are related approaches to discovering logical patterns within data sets. Although rules and decision trees may seem similar at first, they are in fact quite different both in terms of the information they discover from databases and in terms of their behavior on new data items. In essence, decision trees may be viewed as a simplistic approach to rule discovery. [17]

In each case rules will win over trees because they are based on a more sophisticated form of discovery. The fundamental problems with decision trees are at least four:

- They look at very simple combination of attributes within a table, and hence miss many patterns.
- By their very nature, they need to break numeric fields into fixed ranges, hence missing even more patterns, and providing less information. They are quite brittle on inexact data, and a small change in a value can have a large impact on the outcome.
- Decision trees can at best work on small samples of data and can not easily approach large data sets, resulting in significant loss of information.
- Since they ignore some attributes, they may make less accurate predictions, and if some values are missing from the new data item, they make no predictions at all. Furthermore, given the same data set, one can obtain several decision trees, each making a different prediction on new data items!

Misses of decision trees

1- Any decision tree can be expressed as a set of rules, but there are sets of rules, which can not be expressed as any single decision tree.
2- Rules are more expressive than trees on numeric values. There are an infinite number of patterns in data sets, which can not be discovered by decision trees with a fixed assignment of ranges to numeric fields, but which can be discovered with rules. Decision trees thus miss many numeric patterns that rules can find.
3- The ability of decision trees to deal with a good number of non-numeric values is significantly limited when applied to large data sets. More over, trees are usually very hard to understand. Hence the ability of decision trees to produce understandable and readable information from large data sets is significantly ineffective
4- Rules are also better at making predictions. The first problem trees encounter in prediction has to do with the combination of values, e.g. in some cases depending on which decision tree we have formed we will get different answers for prediction.
5- Rules are better in dealing with real world data, which sometimes has missing values. Trees will not be able to make predictions when unknown, missing or even wrong values are encountered in the data. This type of "fragility" is quite serious in real world applications

2.3.2.4 Neural networks classification

Neural Networks have a remarkable ability to derive meaning from complicated or imprecise data and can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an “expert” in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer “what if” questions.

Neural networks have been used successfully for classification where classification is a major problem in data mining. Symbolic classification rules can be discovered using neural networks; however, the general impression is that neural networks are not well suited for data mining. This is due to three facts, which are:

1. The learning time of the neural net is usually long.
2. The classification process may be lost in the graph and the weights assigned to the links between nodes in the neural net.
3. Available domain knowledge is difficult to be incorporated to a neural network.

On the other hand, different from the ideas that excludes the neural network based approaches, neural network approaches should have their position in data mining because of its merits such as low classification error rates and robustness to noise. One major neural network based data mining approach consists of three main phases:

1. Network construction and training: this phase constructs and trains a multi-layer neural net based on the number of attributes and number of classes.
2. Network pruning: the pruning phase aims at removing redundant links and units without increasing the classification error rate of the network.
3. Rule extraction: this phase extracts the classification rules from the pruned network.[18]

2.3.3 Sequential and temporal Discovery

Sequential or temporal pattern discovery is the detection of patterns where the presence of some items in transactions is followed by the presence of other items in other transactions over a time period. (E.g. people who buy shirts will buy ties over a certain period of time). Graphical techniques are used in link analysis for *similar time sequence discovery*, that is, finding sequences similar to a certain sequence. For instance, when used in stock market analysis, this technique could help to relate stocks price behavior to sequences of values of market variables over time.[19]

In the association discovery discussed above, the identity of the customer that did the purchase is not generally known. If this information exists, an analysis can be made of the collection of related records of the same structure as above (i.e., consisting of a number of items drawn from a given collection of items). The records are related by the identity of the customer that did the repeated purchases.[13]

These functions analyze a set of records over a certain period of time to identify given trends. A sequential pattern function will analyze a collection of interrelated records with frequently occurring patterns over time. For example, it can be applied to detect powerfully a set of customers associated with certain kinds of accidents.

2.3.4 Conceptual Clustering and Segmentation

The process of grouping physical or abstract objects into classes of similar objects is called clustering or unsupervised classification.[3] Also, Clustering and segmentation in databases are the processes of separating a data set into components that reflect a consistent pattern of behavior. Once the patterns are established, they can be used to deconstruct data into more understandable subsets and provide subgroups of the population for further analysis of large databases.

First, the system is to discover a subset of related objects in the training set i.e. database, and then it has to find descriptions, which describe each of these subsets. This is a process of partitioning the database into classes of records, which are similar according to certain criteria.

A cluster is the set of records or objects grouped together because of this proximity or similarity. Objects are often decomposed into an exhaustive and/or mutually exclusive set of clusters. The key to clustering according to similarity is to translate intuitive measures into quantitative ones following one of two approaches:

1. Form rules, which dictate membership in the same group, based on the degree of similarity between members.
2. Build a set of functions that measure some property of partitions as functions of some parameters of the partition.

The results of a clustering operation are used in one of two ways. First, for summarizing the contents of the target database by considering the characteristics of each created cluster rather than those of each record in the database. Second, as an input to other methods, e.g., supervised induction. A cluster is a smaller and more manageable data set to the supervised inductive learning component.[13]

As a data mining task, data clustering identifies clusters in densely populated areas, according to some distance measurement, in data or object sets. In machine learning, cluster analysis often refers to *unsupervised learning*, since which class an object belongs to is not pre-specified, or *conceptual clustering*, since the distance measurement may not be based on geometric distance, but be based on that a group of objects represents a certain conceptual class.[3] Measures of similarity between the objects should be defined before applying it to the data sets to determine classes or clusters. Classes are defined as collections of objects whose intraclass similarity is high and interclass similarity is low. Distance or dissimilarity measures are of different kinds and are selected according to application needs. The most popular are vector or square distance, Haming distance, and Bayesian theorem.[20]

The *Squared distance* or *Chi-square* is the distance between two vectors. In clustering this measure is used to compute the distance between a tuple t_i, represented as a vector $(x_{1i}, x_{2i}, \ldots x_{mi})$ and a centroid vector $(y_1, y_2, \ldots y_m)$, that indicates the average characteristics of the concerned group. The "squared distance" is defined as

$$D^2_{ik} = (x_{1i} - y_1, x_{2i} - y_2, \ldots, x_{mi} - y_m) C_k^{-1} (x_{1i} - y_1, x_{2i} - y_2, \ldots, x_{mi} - y_m)^T$$

Where C_k^{-1} is the inverse covariance matrix of the kth group and $(x_{1i} - y_1, x_{2i} - y_2, \ldots, x_{mi} - y_m)^T$ is the transpose matrix of $(x_{1i} - y_1, x_{2i} - y_2, \ldots, x_{mi} - y_m)$.

The *Haming distance* is the measure of dissimilarity between two tuples. The haming distance is defined as

$$D(r, s) = \Sigma_i{}^m{}_{=1}\, f(a_{ri}, a_{si})$$
$$f(a, b) = 0 \text{ if } a = b$$
$$= 1 \text{ otherwise}$$

D(r, s) is the measure of the distance or dissimilarity. The goal of the algorithm is the total hamming distance. Which is the summation of the haming distance between all pairs of consecutive records.

2.3.5 Visualization

Data mining necessitates the use of interactive techniques that allow the user to quickly and easily interpret, analyze, and change the information displayed.

Visualization techniques are among the most powerful devices that help in identifying hidden patterns in data. Features that are difficult to be detected by scanning rows and columns of numbers in databases often become obvious when viewed graphically. They are particularly useful for detecting phenomena that holds for a rather small subset of the data, and go undetected when statistical techniques are used. Therefore, visualization techniques are useful for detecting deviations, while statistics are used to measure their significance.[13]

Even though visualization does not automatically extract information, it facilitates the user in identifying patterns hidden in data, as well as in better comprehending the information extracted by other techniques. The advantage of using visualization is that the analyst does not have to know what type of phenomenon he is looking for in order to notice something unusual or interesting.

Visualization of data with some inherent two or three-dimensional semantics has been done even before computers were used to create visualizations. Since computers are used to create visualizations, many novel techniques have been developed and existing ones have been extended to work for larger data sets and make the displays interactive. Following are some important techniques, which are suitable for visually mining large databases.[21]

2.3.5.1 Pixel-Oriented Techniques

The basic idea of pixel-oriented techniques is to map each data value to a colored pixel and present the data values belonging to one attribute in separate windows. Pixel-oriented techniques use different arrangements for different purposes. Two kinds of arrangement techniques are known query-independent and query dependent.

Query-independent visualization technique is used if the aim is to visualize a large data set. This technique sorts the data according to some attribute(s) and uses a screen filling pattern to arrange the data values on the display. It is especially useful for data with natural ordering according to one attribute.

Query-dependent visualization technique is used if the aim is to visualize the data in the context of a specific user query. If there is no natural ordering of the data and the main goal is an interactive exploration of the database, the user will be more interested in the feedback to some query. Here the use of query dependent technique, which visualizes the relevance of the data items with respect to a query, is appropriate. Instead of directly mapping attribute values to colors, the distances of attribute values to the query are mapped to colors.

2.3.5.2 Geometric Projection Techniques

Geometric projection techniques aim at finding "interesting" projections of multidimensional data sets. The class of geometric projection techniques includes techniques of exploratory statistics such as principal component analysis, factor analysis and multidimensional scaling, many of which are subsumed under the term "projection pursuit". One of the most important geometric projection techniques is the parallel coordinate visualization technique. The parallel coordinate technique maps the *n*-dimensional space onto the two display dimensions by using *n* equidistant axes that are parallel to one of the display axes. The axes correspond to the dimensions and are linearly scaled from the minimum to the maximum value of the corresponding dimension. Each data item is presented as a polygonal line, intersecting each of the axes at that point which correspond to the value of the considered dimension.

2.3.5.3 Icon-Based Techniques

The idea in icon-based techniques is to map each multidimensional data item to an icon. First approaches of iconic displays are Chernoff faces. But the number of data items that can be visualized using the Chernoff face technique is limited. Stick Figure technique allows visualization for larger amounts of data and thus is more adequate for data mining. In the stick figure technique the icon is some type of stick figure. Two dimensions are mapped to two display dimensions and the remaining dimensions are mapped to the angles and/or limb lengths of the stick figure icon. Different stick figures with different dimensionality may be used. Note that in both, the stick figure and Chernoff face technique, the number of dimensions that can be visualized is limited. Shape coding approach allows the visualization of arbitrary number of dimensions. The icon used in the shape coding approach maps each dimension to small array of pixels and arranges the pixel arrays of each data item into a square or a rectangle.

2.3.5.4 Hierarchical and Graph-Based Techniques

The hierarchical techniques subdivide the *n*-dimensional space and present the subspaces in a hierarchical fashion. It mainly focuses on visualization of multivariate functions and thus not interesting for data mining. The basic idea of graph-based techniques is to effectively present large using specific layout algorithms, query languages and abstraction techniques.

2.4 *Information Mining Approaches*

The process of extracting knowledge out of information as well as from data is well known as *mining*. Data has in itself very little meaning and is the raw material we get from the sources. It can be a set of discrete facts about events, and in that case, it is most usefully described as *structured* records of transactions, and it is usually of numeric or literal type. But documents and Web pages are also a source of an *unstructured* data, delivered as a stream of bits, which can be decodified as words and sentences of text in a certain language.

Unstructured data is estimated to represent 80% of an enterprise information compared to 20% from structured data. Unstructured data comprises data from different sources, such as text, image, video, and audio. However, text is the most predominant kind unstructured data. The users are "drowning in data and information" while starving for the insights and knowledge that will allow them to make better decisions.

Structured data is the basis of databases. As a result, mining structured data is interchangeably well known as "data mining" or "knowledge discovery in databases". On the other side, unstructured data is mostly textual. For this reason, mining unstructured data is referred by "Text mining".

"Information mining" is the process of extracting knowledge form any source of data or information. Since textual data is usually semantically richer than numeric data because each text document provides information. To access structured data we talk about "data access", while to access a document we talk about "information retrieval". As a result, "Information Mining" comprises both "Data Mining" and "Text Mining".

2.4.1 Text Mining

Text mining involves two aspects: information retrieval, and text analysis. *Information retrieval* systems facilitate users finding the information they need. *Text analysis* tools help extracting key knowledge from text, organize documents by subject, and find predominant themes in a collection of documents using supervised or unsupervised machine learning techniques. They also help the user to express their real knowledge needs, and provide navigational facilities.

One of the key differences between data and text mining approaches to knowledge discovery is due to the inherent differences in the way the sources need to be handled. As we discussed before, structured data has little semantic meaning, and when extracted from a normalized database there is no relationship among the attributes of a record that can be derived from the data itself.

Furthermore, in data mining the data has to pass through several preprocessing phases before being mined. This preprocessing of data does not occur in text mining since the correctness of a document is not an issue, while only its content is. The problem with textual information, however, is that it is not designed to be handled by computers. Unlike the tabular information typically stored in databases today, documents have limited internal structure, if any. Furthermore, the important information they contain is not explicit but is implicit, buried in the text. Text mining includes full text search (e.g. Web search engines), text analysis (e.g. natural language processing), and document query and retrieval.

2.4.2 Data Mining

Structured data was first presented in file based systems. Then relational database systems emerged to replace file-based systems. Although relational database systems solved many file-based problems like redundancy, inconsistency and update anomalies, it is not appropriate for data mining applications. Current relational database systems have been designed and tuned for On-Line Transaction Processing (OLTP) applications. OLTP applications typically automate clerical data processing tasks that are the day-to-day operations. These operations are structured, repetitive, and consist of short, atomic, isolated transactions.

On the other hand, data mining is based on query or analytic processing rather than transaction processing and is mainly targeted for decision support. Consequently, new kind of databases has been designed to support the performance and functional requirement of On-Line Analytic Processing (OLAP). OLAP requires special data organization, access methods, and implementation methods that generally are not provided by commercial DBMSs targeted for OLTP. As a result, multidimensional data models and operations support OLAP, while relational data model and operations support OLTP.

Data mining and decision support systems might be supported by standard or extended Relational DBMSs, called Relational OLAP (ROLAP), as well as by fully multidimensional DBMSs, called Multidimensional OLAP (MOLAP).

2.4.2.1 ROLAP and RDBMS

Relational OLAP is RDBMS based with extensions to support data mining requirements. In ROLAP, data is stored in relational databases but they support extensions to SQL and special access and implementation methods to efficiently meet the multidimensional data model.

Entity Relationship diagrams and normalization techniques are popularly used for database design in OLTP environments. However, the database designs recommended by ER diagrams are inappropriate for decision support systems where efficiency in querying and in loading data are important. Consequently, new structures like Star schema, Snowflake schema, and fact constellation, are used in ROLAP to represent the multidimensional data model. [22]

Star schema: the database consists of a single *Fact* table and a single table for each dimension. Each tuple in the fact table consists of a pointer (foreign key) to each of the dimensions that provide its multidimensional coordinates and store the numeric measures for those coordinates. Each dimension table consists of columns that correspond to attributes of the dimension.

Snowflake schemas: provide a refinement of star schema where the dimensional hierarchy is explicitly represented by normalizing the dimension tables. This leads to advantages in maintaining the dimension tables. However, the denormalized structure of the dimensional tables in star schemas may be more appropriate for browsing the dimensions.

Fact constellations: are examples of more complex structures in which multiple fact tables share dimensional tables.

In addition to the fact and dimension tables, ROLAP databases store selected summary tables containing pre-aggregated data, which corresponds to aggregating the fact table on one or more selected dimensions.

2.4.2.2 MOLAP and DATA CUBES

Multidimensional OLAP (MOLAP) refers to OLAP implemented using hypercubes to represent the data as a multidimensional structure. Hypercubes are data objects defined by dimensions where dimensions are the heterogeneous objects that provide categories for data. As mentioned before, ERD provides an inappropriate way of modeling hypercubes. ERD models give no understanding of the processing logic stored as database objects and assume that the data is operated on by programs, and therefore represent only the relationships between objects. [23] On the other side, multidimensional model stresses on aggregation of measures by one or more dimensions as one of the key operations and comparing two measures aggregated by the same dimensions.[22]

Determining attributes like *product, date, market* are refered to as *dimensions* while the determined attributes like *sales* unit values are referred to as *measures*. Usually dimensions are called categorical attributes and measures are called numeric or summary attributes. [24]

The dimensions together are assumed to uniquely determine the measure. Thus, the multidimensional data views a measure as a value in the multidimensional space of dimensions. Each dimension is described by a set of attributes. The attributes of the dimension may be related via a hierarchy of relationships.

Dimensions usually are associated with hierarchies that specify aggregation levels and hence granularity of viewing data. For example, day$\rightarrow$ month $\rightarrow$ quarter $\rightarrow$ year is hierarchy on the time period that specifies various aggregation levels.

Some of the popular operations that are supported by multidimensional databases are summarized as follows: [22] [24]

- *Pivoting* can be viewed as the operation of transforming value sets in columns of original tables to row and column headers in the pivoted tables or rotate the cube to show a particular face.
- *Slice-and dice* corresponds to reducing the dimensionality of the data, i.e., selecting some subset of the cube.
- *Rollup* corresponds to taking the current data object and doing further group-by on one of the dimensions
- *Drill-down* is the converse of rollup. It displays detail information for each aggregated point. It is useful because often users want to see only aggregated data first and selectively see more detailed data, in a lower aggregation level, with the help of hierarchies.

2.5 *Repositories*

Data mining and Decision Support Systems require different kind of data stores to handle their different kind of process requirements. Data warehouses and data marts represent

- *Drill-down* is the converse of rollup. It displays detail information for each aggregated point. It is useful because often users want to see only aggregated data first and selectively see more detailed data, in a lower aggregation level, with the help of hierarchies.

2.5 Repositories

Data mining and Decision Support Systems require different kind of data stores to handle their different kind of process requirements. Data warehouses and data marts represent the new kind of data stores. Data stores are not enough and should be accompanied with pattern stores.

2.5.1 Data warehouses

A data warehouse is a "subject-oriented, integrated, time varying, non-volatile collection of data that is used primarily in organizational decision making."[25] Data warehouses are targeted for decision support. Enterprise, historical, summarized and consolidated data is more important than individual records. They contain data from several operational databases, over long periods of time and thus tend to be orders of magnitude larger than operational databases.

"A data warehouse is a dedicated data store that contains multiple years' worth of data sourced from various operational systems"(Wayne Eckerson).

In general a data warehouse can be described as follows:

- Subject oriented: data is organized according to subject instead of application and the data organized by subject contain only data necessary for decision making.
- Integrated with quality data management: data collected from several operational applications must be consistent and assume standard coding convention. Also, the fact-based management demands high data quality.
- Time variant & non volatile: the data is usually historical data and is not updated except on periodical bases where it is only loaded and then accessed.
- Performance: mining and ad hoc analysis processes must be accompanied with high performance over huge amounts of data that serve concurrent users.
- Integrated dimensional analysis: dynamic calculation of aggregates should be compliant to the interactive performance needs. Since the workloads are query intensive with mostly ad hoc, complex queries that access huge amount of records and perform considerable amount of full table scans, joins, and aggregations, throughput and response time is affected by query through put rather than transaction throughput.

2.5.2 Data marts

Unfortunately, Centralized data warehouses have not lived up to the expectations. They have instead proven to be costly, long to implement and filled with risk. Fortunately, a new and more pragmatic approach is now available: Distributed Data Warehousing. The focus in this approach is on applications and Data Marts, built rapidly with new cost-effective, scaleable database technology.

Data Marts can provide dramatic returns on investments with no risk of failure. This enables organizations to rapidly develop Decision Support System (DSS) that can change and grow as requirements evolve. Although, a Data Mart is the store for DSS that focuses on solving a specific business problem in a single department or subject area, it must be built with enterprise data model in mind to prevent complex integration problems on the long run.

There are four key components to a successful data mart strategy:[26]

- Flexible technology for rapid implementation
- Scalability to large data volumes and support for detailed data
- Fast, flexible and scalable complex query and operation response
- Data consistency, transformation, and refresh architecture

2.5.3 Pattern warehouses

In the 1990's it became clear that data in warehouses are often too coarse and unmanageable for detailed decision making -- business users needed much more refined knowledge. Moreover, most organizations realized that what they really wanted was the *knowledge, trends and patterns* within the data -- not the data itself.

A *Pattern Warehouse* ™ is a repository that holds historical patterns rather than historical data. With a pattern warehouse, almost all the relevant patterns in the data are *found beforehand,* and stored for use by business users who get the interesting patterns of change every week or month or can query the Pattern Warehouse at will.

Knowledge is so much more compact than data. As a result, the *Pattern Warehouse* ™ is only a fraction of the size of the data warehouse, allowing the patterns many years to be stored with ease, even when the data is no longer available. This provides a huge amount of knowledge over time at a low cost for disk space.

Response time is far better than the data warehouse because the patterns have already been extracted, ready for look-up. This provides an environment for long term *corporate knowledge management.*

Accessing patterns in the *Pattern Warehouse* requires a new query language that can perform the job. PQL: The Pattern Query Language is the new query language that was designed for that purpose. PQL is very much similar to SQL. While SQL relies on relational algebra, PQL uses "pattern algebra".[27]

Response time is far better than the data warehouse because the patterns have already been extracted, ready for look-up. This provides an environment for long term *corporate knowledge management*.

Accessing patterns in the *Pattern Warehouse* requires a new query language that can perform the job. PQL: The Pattern Query Language is the new query language that was designed for that purpose. PQL is very much similar to SQL. While SQL relies on relational algebra, PQL uses "pattern algebra".[27]

The knowledge access paradigm is a *truly revolutionary idea* with a multitude of business and technical benefits that reinforce each other. To get a sense of the benefits, consider the following comparison.[27]

The Way It Was	*The Way It Is*
Without a PatternWarehouse	With a PatternWarehouse
• Users had to manipulate raw data to find patterns with substantial effort.	• Patterns are found beforehand, users just access them with great ease. No training.
• Users needed training, or had to rely on analysts for pattern discovery.	• Users just click on a graphic user interface for pattern query.
• Pattern analysis was adhoq and fragmented, results often varied from user to user.	• Patterns are stored in a central repository. All business users get uniform answers.
• Analytical reports were cryptic and hard to understand. Often no explanations.	• Reports are in plain English text with graphs automatically generated on the Intranet.
• Turn-around time for follow up questions was long after a first analysis.	• Turn-around is instant because patterns are just looked up, not re-computed each time.
• Analysis was performed on extract files. Patterns were missed or were unreliable.	• The entire database is analyzed. Powerful and accurate patterns are found.

Chapter3

The Original Metapattern Generator

3.1 Introduction

This area of data mining focuses on discovering relation-based patterns from data in a structured format. Most of the existing algorithms focus on deriving new predicates from positive and negative training examples. Classification algorithms usually generate decision trees instead of implication rules. Supervised learning needs extensive interference from the user or the teacher. Although this kind of algorithms was applied and used in more than one domain and generated acceptable results, they have many drawbacks as they require considerable user interference and lack proper Knowledge representation as they generate structures that are hard to understand in addition to being too time consuming. The new approach focuses on minimizing the user interference and generating interesting Value Sensitive rules. The algorithm, as will be shown, extends applicability to intelligent retrieval planning and record/case clustering in record/case-based systems. To achieve this, we make use of the unsupervised learning and integrated knowledge discovery systems. Integrated knowledge discovery systems integrates Induction, Deduction, and human external knowledge in an iterative discovery loop.

3.2 The original algorithm

The metapattern generation system [28] is an integrated data mining system that integrates deduction, induction and human guidance. Although the metapattern-based human-directed discovery loop has already been successfully applied for Knowledge discovery, an automated discovery loop is needed to give human users suggestions for new metapatterns. Data mining techniques like Induction, Deduction and human guidance are intrinsically interdependent. In order to exploit these interdependencies they have to be integrated in an automated discovery loop. In the deductive part of the loop, metapatterns outline the data-collecting strategy and serve as the basis for the generation of specific queries. In the inductive part of the loop, metapatterns serve as generic description of the class of patterns to be discovered. More expert users can get inspirations out of metapatterns, and the less expert users can learn how to perform data mining in a particular domain by observation. The integration of different data mining techniques has become a necessity for it improves both the results and performance.[28]

3.3 The metapattern and Compute Strength

Metapattern is proposed as a template or a second order expression in a language *(L)* that describes a type of pattern to be discovered. A metapattern is a generalization of similar patterns. It is expressed in form of predicates where predicate names generalize table names and predicate parameters generalize attribute names. For example the metapattern

$P(X, Y) \wedge Q(Y, Z) \Rightarrow R(X, Z)$,

specifies that the patterns to be discovered are transitive. The result of executing a metapattern is a set of patterns whose left-hand sides are instantiated forms of the left-hand side of the metapattern, and whose right-hand sides are the results of the corresponding metapattern action.

In fact the more precise formulation of the above metapattern is:

$P(X, Y) \wedge Q(Y, Z) \Rightarrow \text{CompStrength}(R(X, Z))$

CompStrength(R (X, Z)) is an action that computes the strength of the pattern. Each pattern is evaluated against the database U_{db} by two values:

The Strength value P_S, which is the probability of seeing the right hand side of p being true when the left-hand side of p is true. Based on "Laplace's Rule of succession" [29], the strength value of *p* can be computed as:

$p_s = Prob(RHS|LHS, U_{db}, I_O) = (|S_{RHS}|+1) / (|S_{LHS}|+2)$

The Base value P_b, which estimates how likely the left-hand side of p occurs in databases that have the same schema of U_{db} is estimated by the following formula:

$p_b = (|S_{LHS}|) / DOM(\text{LHS})$

where LHS represents the left-hand side of *p*, RHS the right-hand side of *p*, S_{LHS} the set of tuples in U_{db} that satisfy LHS, S_{RHS} the set of tuples in S_{LHS} that satisfy RHS, *DOM* (LHS) is the product of sizes of the tables that appear in LHS and I_O is the assumption that the prior distribution of S_{RHS} in S_{LHS} is uniform. Intuitively, this p_s value is the probability of seeing a tuple that satisfies RHS given the condition that the tuple satisfies LHS.[28]

A pattern is said to be "interesting" only if its strength is within the user-specified threshold. It is said to be plausible if base is above the user-specified threshold. When $p_s \geq s$ or $p_s \leq 1\text{-}s$, the pattern is accepted. Otherwise, if the base value is still above its threshold (i.e., $p_b \geq b$), then the pattern is considered plausible.[28]

3.4 The Automated discovery loop

The generation of metapatterns by users is very difficult even for the most expert users. If a metapattern is too specific, it may miss the interesting patterns. If it is too general, it may exhaust the computing resources that are available. To aid the user in the pattern generation process, a metapattern generator was put in parallel with the user. Based on data, the generator will generate the metapatterns out of which patterns are generated. Humans will calibrate the generated metapatterns based on their external knowledge.[28] The system should provide suggestions and feedback of metapatterns so the experts can discover new knowledge and try better metapatterns. For this purpose, the metapattern generator for the metapattern-based discovery loop was developed.[30]

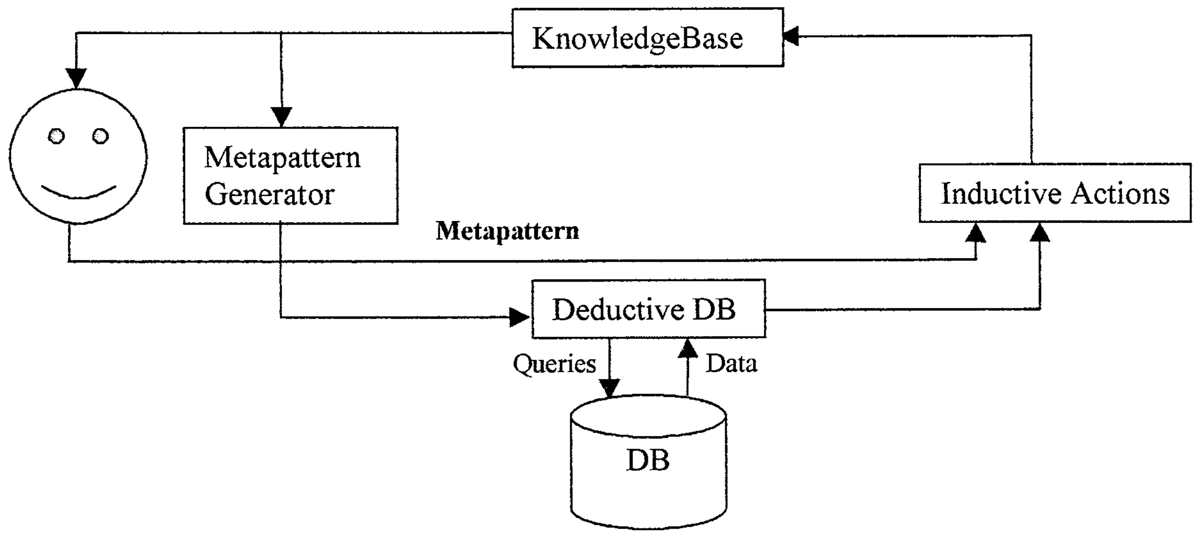


Figure 3.4.1 the automated discovery loop with a metapattern generator[28]

3.5 *The metapattern Generator.*

The original metapattern generator looks for overlaps between attributes of different tables. The overlap is measured in terms of finding the ratio of intersection of values in the underlying database of the two columns. Given two columns C_x and C_y, the overlap is defined as follows: [30]

Overlap $(C_x, C_y) = \max(|V_x ? V_y| / |V_x|, |V_x ? V_y| / |V_y|)$, Equation 1

where V_x and V_y are the value sets of C_x, and C_y respectively. If the computed overlap is greater or equal to user defined threshold, then a reference is created for the two columns and inserted to a table called a Significant Connection Table (SCT). The SCT is then tested for finding cycles with alternating edges. Out of the found cycles the transitive metapatterns are generated. The algorithm can be outlined as follows:

```
Procedure MetaPatternGenerator (D as DataBase, S as Schema, o, b, s as Threshold)
  D is the underlying database
  S is the schema of the database
  o, b, and s are the user defined overlap, base and strength thresholds respectively
     For each two columns $C_x$ & $C_y$ from different tables
        If $C_x$ & $C_y$ are of the same type then
        OL =Overlap ($C_x$, $C_y$)
        If OL >= o then
             Add the term of these attributes to Significant Connection Table
     End For
     Find Cycles in the graph that can be generated from the Significant Connection Table.
     Generate MetaPatterns from the generated cycles.
End.
```

Figure 3.5.1 The metapattern generator

Generating all transitivity metapatterns is not the end of the story. When a metapattern is selected, the system first instantiates it into a set of specific patterns that are possible in the current databases. Patterns are found by instantiating predicates by table names and predicate parameters by attribute names. Notice that not all instantiated patterns are supported by the underlying database. A pattern is interesting only if its significance is above a user defined threshold[30]. As a result, each pattern is evaluated against the database by computing the *Strength* P_s and *Base* value P_b defined in section 3.3, which are compared with two user specified thresholds *b* and *s*. If the base and strength values are both above the user-defined thresholds, the pattern is accepted. If only the base value is above the threshold, the pattern is considered plausible.

As observed in [28], domain experts can make the entire discovery process much more efficient and productive if they interact directly with the system. For this purpose, the metapattern generator is integrated in an automated discovery loop, as shown in Figure 3.4.1. The technique is supported by human interaction to add more credibility to the generated patterns. Humans can interact with the generator by examining, selecting, and executing the metapatterns, and they can create their own metapatterns as well. The algorithm for the pattern generation in the automated discovery loop can be outlined as follows:

```
Procedure PatternGenerator (s,b as threshold)
M: = MetaPatternGenerator (D,S,o,b,s)
Loop
  Order and Display M;
  Let User examine, create, and reorder M;
  Select m from M;
  For each Pattern p instantiated from m;
          Compute the strength value Ps and the base value $P_b$;
          If $P_s$ >= s or Ps <= 1-s then
                  Output (p);
          Else if $P_b$>= b then
      Select a set C of constraints for p;
                  Create new metapattern by adding the constraint to the left-hand side of p;
                  Insert the new mettapattern to M;
Until M is empty or the user instructs to stop;
```

Figure 3.5.2 The control algorithm of the automated discovery loop

The following database example is taken to describe the metapattern discovery process. The metapattern generator as described before will search for the overlapping attributes from different tables and create a reference name for this connection. The reference name is inserted into a table called the Significant Connection Table.

Table T_1		
c11	c12	c13
jj	5	0.5
nn	5	0.8
ll	7	0.5
qq	5	0.5
kk	5	0.6
pp	4	0.6
mm	2	0.5
nn	4	0.6
kk	4	0.4
nn	5	0.4

Table T_2		
c21	c22	c23
14	0.5	mm
14	0.6	iii
14	0.3	jjj
12	0.7	nnn
12	0.1	lll
15	0.6	ppp
15	0.4	mm
13	0.6	ooo
16	0.6	ooo
17	0.4	mm
14	0.4	lll
14	0.6	kkk
15	0.3	mm
12	0.5	mm
15	0.4	nnn
15	0.6	ooo
16	0.5	ppp
16	0.7	ppp

Table T_3	
c31	c32
14	oo
15	kk
16	mm
15	kk
16	ll
15	ll
15	mm
13	oo
16	oo
14	mm
13	mm
14	mm

Table T_4		
c41	c42	c43
nnn	0	5
mm	0	6
rrr	0.1	4
mm	0	4
ooo	0.1	7
jjj	0	4
kkk	0	5
mm	0	5
jjj	0.1	4
mm	0	5
jjj	0	5
jjj	0	5
lll	0	7
nnn	0.1	4

Figure 3.5.3 An Example Database

By applying the metapattern generator in Figure 3.5.1 to the above database example, each pair of columns that are connected are then given a reference name X_i and added to the

Significant Connection Table (SCT). As a result, the following SCT is generated from the database Example in Figure 3.5.3.

SCT

	T_1	T_2	T_3	T_4
X_1	C_{13}	C_{22}		
X_2	C_{11}		C_{32}	
X_3	C_{12}			C_{43}
X_4		C_{21}	C_{31}	
X_5		C_{23}		C_{41}

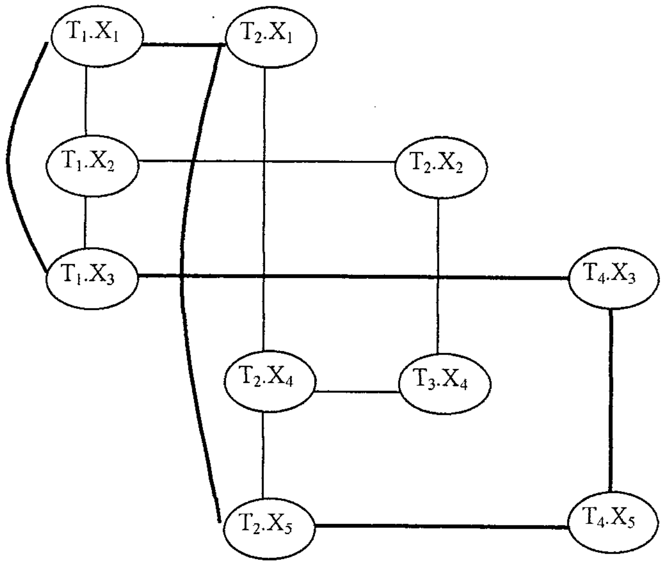


Figure 3.5.4 A Significant Connection Table (SCT) and its graph G. [30]

Suppose the threshold *o* for overlapping is set to 0.6, then column $T_1.c_{13}$ and $T_2.c_{22}$ are connected because they are defined over the same domain and their overlap value is 0.9. A reference name , X_1, is then created for this pair of connected columns. After considering every pair of columns of the same type in the example database, the Significant Connection Table in Figure 3.5.4 is constructed. The SCT is represented in a graph *G*, where each node in *G* is a non-empty entry in SCT, and each edge connects two non-empty entries that are on the same row or column in the SCT. The idea is to find cycles in the graph with alternating vertical and horizantal edges, and convert each of these cycles into a "cycle of predicates" before generating a set of transitivity patterns. For example, the cycle indicated by thick lines in Figure 3.5.4 is (T_2, X_1) (T_2, X_5) (T_4, X_5) (T_4, X_3) (T_1, X_3) (T_1, X_1) (T_2, X_1) and as a cycle of predicates as "$T_2 (X_1, X_5)$ $T_4 (X_5, X_3)$ $T_1 (X_3, X_1)$". From this predicate, the following patterns can be generated:

$T_2 (X_1, X_5)\ T_4 (X_5, X_3) \rightarrow T_1 (X_1, X_3)$

$T_1 (X_3, X_1)\ T2 (X_1, X_5) \rightarrow T_4 (X_3, X_5)$

$T_4 (X_5, X_3)\ T_1 (X_3, X_1) \rightarrow T2 (X_5, X_1)$

All the cycles that are generated from the graph *G* in Figure 3.5.4 are the following;

$T_2 (X_1, X_5)\ T_4 (X_5, X_3)\ T_1 (X_3, X_1)$

$T_2 (X_1, X_4)\ T_3 (X_4, X_2)\ T_1 (X_2, X_1)$

$T_2 (X_5, X_1)\ T_1 (X_1, X_2)\ T_3 (X_2, X_4)\ T_2 (X_4, X_5)$

$T_2 (X_4, X_5)\ T_4 (X_5, X_3)\ T_1 (X_3, X_1)\ T_2 (X_1, X_4)$

$T_1 (X_3, X_1)\ T_2 (X_1, X_4)\ T_3 (X_4, X_2)\ T_1 (X_2, X_3)$

$T_3 (X_2, X_4)\ T_2 (X_4, X_5)\ T_4 (X_5, X_3)\ T_1 (X_3, X_2)$

$T_1 (X_2, X_3)\ T_4 (X_3, X_5)\ T_2 (X_5, X_1)\ T_1 (X_1, X_2)$

$T_2 (X_1, X_4)\ T_3 (X_4, X_2)\ T_1 (X_2, X_3)\ T_4 (X_3, X_5)\ T_2 (X_5, X_1)$

$T_1 (X_1, X_2)\ T_3 (X_2, X_4)\ T_2 (X_4, X_5)\ T_4 (X_5, X_3)\ T_1 (X_3, X_1)$

From the above set of cycles the following set of generalized transitivity metapatterns are generated

$P_1(Y_1, Y_2) \wedge Q_1(Y_2, Y_3) \rightarrow R_1(Y_1, Y_3)$
$P_2(Y_1, Y_2) \wedge Q_2(Y_2, Y_3) \wedge W_2(Y_3, Y_4) \rightarrow R_2(Y_1, Y_4)$
$P_3(Y_1, Y_2) \wedge Q_3(Y_2, Y_3) \wedge W_3(Y_3, Y_4) \wedge V_3(Y_4, Y_5) \rightarrow R_3(Y_1, Y_5)$

3.6 *The computational complexity of the MetaPattern Generator*

We attempt here to estimate the computational effort of the above algorithm. First, computation cost of the overlap defined in Equation 1 above is O (p^2), where p is the cardinality of the table instance. Since, we have n^2 overlaps to be computed, where n is the number of attributes, it follows that the time complexity of computing all overlaps is O (p^2n^2).

After generating the SCT or the graph, the algorithm looks for cycles in the graph.The graph generated has at most n^2 nodes. Finding cycles in the graph can be solved either by Warshall's or Strassen's algorithm. Warshall's algorithm solves the problem in O (m^3) Strassen's algorithm can be used to solve the problem in O $(m^{2.81})$ where m is the number of nodes by using a set of arithmetic operations [31].

In this case the number of nodes in the graph is n^2, where n is the number of attributes. When substituting m for n^2, we obtain the time complexity of finding cycles in terms of attributes. Thus the problem is solved in O $(n^{2*2.81})$ or O $(n^{5.62})$. This means that the computational complexity of generating metapatterns is O (p^2n^2) + O $(n^{5.62})$

The maximum number of metapatterns that can be generated from n attributes is n^2 and the maximum length of a metapattern is n^2. From each metapattern of length m we can generate m patterns. This means that from a metapattern of length n^2, we can generate n^2 patterns. From n^2 metapatterns, the maximum number of patterns that can be generated is n^4.

Each pattern of length L requires L joins to test its strength. The length of the pattern is the same as its metapattern length. This means that the pattern length L is n^2. Since the join operation is recursive over the whole pattern, the cost of these joins is p^{n^2}. The worst case has an order of $n^4p^{n^2}$

Therefore, the computational complexity of whole algorithm is given by:

$$O(p^2n^2) + O(n^{5.62}) + O(n^4p^{n^2}) = \text{Max}(O(n^4p^{n^2}), O(n^{5.62}))$$

Chapter4

The New Algorithm

In this chapter we focus on deriving a new metapattern generation algorithm. The original algorithm can be enhanced in both performance efficiency and in the quality of the generated metapatterns. The main observation is that the traditional algorithm does not make use of domain knowledge and is incapable of generating patterns over subdomains. It only generates transitivity patterns that hold over the whole underlying domain. Furthermore, the algorithm in its present form is impractical and does not depart from the realms of theoretical presentation. Its impracticality is partly because it is incapable of updating previously generated metapatterns. Another drawback is that the evaluation of the patterns against the underlying database mainly relies on carrying out joins between several tables. This threatens seriously the practical feasibility of the technique as the database increases in size.

In the new enhanced version, new techniques are applied to improve the performance as well as the quality of generated rules. The newest trend in knowledge discovery is incremental learning. As patterns generated should always reflect the underlying data, the patterns should be updated as the underlying data changes rather than regenerated. Regenerating patterns requires the algorithm to run over the database after each set of updates. Regeneration of patterns is time and resource consuming if the algorithm fails to make use of previously generated patterns. The previous algorithm does not take into consideration incremental generation of metapatterns. We can overcome this problem by applying new techniques for incremental computation of the overlap as well as using the Bayesian approach for the incremental computation of the pattern strength. This is described in details in section 4.7

The new algorithm is capable of generating rules that the original algorithm is not able to generate. Transitivity patterns are not the only interesting kind of patterns that can be generated. By incorporating the domain knowledge and exploiting existing data dependencies, we are able to generate in addition to transitivity patterns, pseudo-transitive patterns, additive patterns and extended transitive patterns.

Since patterns can be defined over sub-domains rather than over a whole domain, we make use of value sensitive dependencies and incorporate the idea of finding overlaps in sub domains to generate value sensitive or class based patterns.

Not only Domain knowledge is used in our new approach to generate previously non-generated patterns, but also it is used to deduce overlaps. Given the Foreign dependencies, we can deduce some overlapping attributes without computing the overlap. This is explained in details in section 4.1.

In the original algorithm each pattern is evaluated against the underlying database. This evaluation can be achieved by joining tables instantiated in the pattern over the overlapping attributes. To minimize this cost, we have developed a new algorithm that reduces vastly the high cost of joining tables with very large instances. The algorithm is described in details in section 4.6

The selection of attributes is a vital issue in inductive learning. In the traditional algorithm, the only criterion for selecting an attribute is to have an overlap with another attribute from a different table. This kind of selection of attributes can result in patterns with no predictive value. In our approach, for an attribute to be selected, it should be interdependent with another attribute in addition to the overlap. Our rules are represented in second order predicate logic, where the predicate parameters are instantiated with attribute names or values. Since, in general, predicate parameters are interrelated and this interrelation is represented by the predicate name, we can say that the selected attributes should be interdependent. This issue is described in details in the following sections.

4.1 *Induced Dependencies*

Induced dependencies can be defined as functional dependencies that are consistent with the data currently held in the database but which have not been defined within the database schema. Induced dependencies are value sensitive and are true based only on the underlying data. In other words, they are true reflections of the data and any changes in the data will affect the previously generated induced dependencies. In the following sections we will present different kinds of induced dependencies.

The notational conventions adopted are based on Maier[32]. A relational schema R is a set of attribute names $\{A_1, \ldots, A_n\}$ where each A_i, $1 \leq i \leq n$, has a corresponding D_i, called as domain of A_i. A tuple t over R is an element of $D_1 \times D_2 \times \ldots \times D_n$. a relation r over R is a finite set of tuples defined over R. If r is a relation on R, with X and Y subsets of R then the Functional Dependency $X \rightarrow Y$ is satisfied by r if and only if for every X-value x, $\pi_Y(\sigma_{X=x}(r))$ has at most one tuple, where π and σ are the project and select operators as given by Maier [32].

Functional Dependency (FD) can be also defined as follows: *Let r be any relation instance on scheme $R(A_1, \ldots, A_n)$, U be the universal set of attributes $A_1, \ldots, A_n$, and both X and Y be subsets of U. Relation instance r is said to satisfy the functional dependency (FD) $X \rightarrow Y$ (reads as "X determines Y") if, for every pair of tuples t_1 and t_2 in r, $t_1[X]=t_2[X]$ implies $t_1[Y]=t_2[Y]$.*

4.1.1 Full Induced Dependencies

Full Induced Dependency is defined as follows: [34]

Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A full induced dependency between X and Y exists in r if:

1) *r satisfies* $X \rightarrow Y$, *i.e.,. the data in r is consistent with the existence of the a functional dependency* $X \rightarrow Y$,
2) $X \rightarrow Y$ *is not implied by the set of FDs which apply to R.*

4.1.2 Weak Induced Dependencies

The absence of full induced dependencies does not mean the absence of an interesting correlation between values. This is the case in data mining where there is a large amount of data and most of the dependencies that can be induced are weak. The measure of weakness of a pattern is the indicator of interestingness of the pattern. Weak induced dependencies are defined as follows: [34]

Let R be a non-empty relation r defined on R and let $X \rightarrow Y$ (k) *exists if for some arbitrary confidence value k', k'≤ k, it is possible to write* $X \rightarrow^{-} Y$. *where k is the measure of weakening.*

4.1.3 Value Sensitive Induced Dependencies

The Value Sensitive Induced Dependency extends further the concept of induced dependencies. In some cases, the full induced dependency does not hold over the whole domain while it might hold in a subset of that domain. In other words, only a subset of values in a set of attributes infers values in a second set of attributes. Value Sensitive Induced Dependencies are defined as follows: [34]

Let r be a nonempty (base or view) relation defined on R and let X and Y be a subset of R. Let D_x *and* D_y *be the domains of X and Y, respectively, and let* D'_x *be a proper subset of* D_x *and* D'_y *be a proper subset of* D_y. *A value sensitive induced dependency between X and Y exists with respect to* D'_x *and* D'_y *if the induced dependency* $X' \rightarrow^{-} Y'$ *holds in r' where* $r' = \sigma_{x \in D'_x \wedge y \in D'_y}(r)$ *and r' is nonempty.*

Three comments should be made about the above definition.

1) When D'_x is equal to D_x and D'_y is equal to D_y then the Value Sensitive Induced dependency simple becomes a Full Induced Dependency, as the select operations specify no restriction.
2) Restricting the domain of X does not require a corresponding restriction in the domain of Y and vice versa.
3) A qualitative measure of the weakening of the Full Induced Dependency is the extent to which D'_x and D'_y are reduced relative to D_x and D_y.

4.2 *Fuzzy Functional Dependency*

Fuzzy Functional Dependency (FFD) [36] is a generalization of Functional Dependency (FD). FFD uses similarity instead of strict equality. As a result, FFD turns into if $t\,[X]$ is similar to $t'\,[X]$ then $t\,[Y]$ is also similar to $t'\,[Y]$. The similarity between Y values is greater or equal to the similarity between similarity between X values. For example, "employees with similar experiences *must have* similar salaries."

FFDs can be also weak dependencies. For example, "the intelligence level of a person *more or less* determines the degree of success."

Fuzzy Functional Dependency is the defined as follows:[36]

Let r be any fuzzy relation instance on scheme $R\,(A_1,\ldots,A_n)$, *U be the universal set of attributes* $A_1,\ldots,A_n$, *and both X and Y be subsets of U. fuzzy relation instance r is said to satisfy the fuzzy functional dependency (FFD)* $X \rightarrow_F Y\,[\theta]$ *if for every pair of tuples* t_1 *and* t_2 *in r,*

$$C\,(Y\,[t_1, t_2]) \geq \min\,(\theta, C\,(X\,[t_1, t_2]))$$

Here, θ *is a real number with in the range [0,1], describing the confidence of the dependency.*

C denotes the conformance (similarity) of two tuples in a given attribute.

Conformance of an attribute is defined as follows:

The conformance of attribute A_k *defined on domain* D_k *for any two tuples* t_1 *and* t_2 *present in relation instance r and denoted by* $C\,(A_k\,[t_1, t_2])$ *is given as*

$$C\,(A_k\,[t_1, t_2]) = \min\,\{\min_{x \in d_1}\,\{\max_{y \in d_2}\,\{s\,(x, y)\}\},\ \min_{x \in d_2}\,\{\max_{y \in d_1}\,\{s\,(x, y)\}\}\}$$

where d_1 *is the value of the attribute* A_k *for tuple* t_1, *where* d_2 *is the value of the attribute* A_k *for tuple* t_2, $s\,(x, y)$ *is a similarity relation for values x and y, and s is a mapping of every pair of elements in the domain* D_k *onto interval [0, 1].*

FFDs are used in our approach to regain the loss of information resulting from categorizing numerical attributes like age. By explicitly defining the similarity between age classes after categorizing, we can use FFDs to generate dependencies from these special attributes. As a result, FFDs can be used then in the generation processes of metapatterns like other dependencies.

4.3 Overlap vs Induced Foreign Dependency

In the original algorithm overlap is defined as a measure of connectivity between two attributes from different tables. [30] But based on our observation an overlap between two attributes of the same table is also possible and even we can compute the overlap between attributes of different views of the same table. As a result, an overlap over the same attribute can be also computed. For consistency with Induced dependencies and for the sake of generating patterns of a new nature we found that induced foreign dependency will do a better job in the pattern generation process. The Foreign dependency can be defined as follows:

Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A Foreign dependency $X \rightarrow Y$ *between X and Y exists in r if:*

1) X and Y are defined under the same domain
2) $V_X \subseteq V_Y$. where V_x and V_Y are the value sets of X and Y respectively.

4.3.1 Full Induced Foreign Dependency

Full Induced Foreign Dependency is defined as follows:

Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A full Foreign induced dependency $X \rightarrow^{-} Y$ *between X and Y exists in r if:*

1) *r satisfies* $X \rightarrow Y$, *i.e.,. the data in r is consistent with the existence of the a Foreign dependency* $X \rightarrow Y$
2) $X \rightarrow Y$ *is not implied by the set of Foreign key Dependencies which apply to R.*

4.3.2 Weak Induced Foreign Dependency

The absence of full induced Foreign dependencies does not necessarily mean the absence of a connection between two sets of attributes. This is the case in data mining where there is a large amount of data and a considerable amount of the Foreign dependencies that can be induced are weak. The measure of weakness of a foreign dependency is the indicator of interestingness of the induced foreign key. A Weak induced foreign dependency is defined as follows:

Let R be a non-empty relation r defined on R and let $X \rightarrow Y$ (k) *exists if for some arbitrary confidence value k', k'≤ k, it is possible to write* $X \rightarrow^{-} Y$. *where k is the measure of weakening.* $k = |V_x \cap V_y| / |V_x|$ Equation 2

where |.| is the cardinality, and V_x and V_y are the value sets of X and Y respectively.

4.3.3 Value Sensitive Induced Foreign Dependency

The Value Sensitive Induced foreign dependency extends further the concept of induced foreign dependency. In some cases, the full induced Foreign dependency does not hold over the whole domain while it might hold in a subset of that domain. Value Sensitive Induced Foreign Dependency is defined as follows:

Let r be a nonempty (base or view) relation defined on R and let X and Y be a subset of R. Let D_x *and* D_y *be the domains of X and Y, respectively, and let* D'_x *be a proper subset of* D_x *and* D'_y *be a proper subset of* D_y. *A value sensitive induced Foreign dependency between X and Y exists with respect to* D'_x *and* D'_y *if the induced Foreign dependency* $X' \rightarrow^{-} Y'$ *holds in r' where* $r' = \sigma_{x \in D'_x \wedge y \in D'_y}(r)$ *and r' is nonempty.*

4.4 *Attribute Selection*

The selection of attributes is a vital issue in inductive learning. Since the metapattern generation process is very expensive, we should guide the selection of the attributes. Selection of the attributes will help in two major issues. First, it will reduce the number of attributes that will be included in the discovery process. Second, it will minimize the number of unimportant patterns that will result in patterns with no predictive value. We are very much concerned about response time as well as in the importance or the predictive value of the generated patterns. In the original algorithm, an attribute is selected if it overlaps with an attribute from another table. This is not enough for the selection of an attribute because it does not prevent the selection of unimportant attributes. Moreover, the assumption that the overlapping attributes should be form different tables will add another weakness to the original algorithm, since important attributes of the same table can overlap, as we will show later.

We can overcome the weakness of selecting unimportant attributes by making use of the interdependency measure between attributes. The patterns generated are represented in second order predicate logic. Since predicate parameters are interrelated and thus represent a given logic, attributes by which the predicate parameters are instantiated should be interdependent as well. The predicate that has its parameters instantiated by unrelated attributes has no meaning. Moreover the strength of the dependency between the attributes will reflect the strength of the predicate. In other words, to have a meaningful predicate its parameters and thus attributes should be interdependent.

Interdependency between attributes can be measured by the strength of the induced functional dependency between the two attributes. The more the strength of the induced dependency, the more the attributes are interdependent. Based on our observation, also attributes in an all key table are important once instantiated in a predicate with the same number of attributes. As a result, we can set the following conditions for the selection of an important attribute:

1- it should have a Foreign interdependency
2- It should have a functional interdependency or a subset of a whole key table.

Given a relation R1 (X1, Y1...), R2 (X2, Y2...), R3 (X3, Y3...), then the two attributes X1 and Y1 are selected if and only if

1- X1→ Y1 or Y1→X1 or (X1, Y1) is the Key
2- X1 overlaps X2 and Y1 overlaps Y3

To achieve the objective of adding only promising or important attributes to the Significant Connection Table, we will look first for attributes in the tables that satisfy condition 1 and we look for overlaps only in this chosen subset of attributes.

Eye Color	Married	Sex	Hair Length	Class
Brown	Yes	Male	Long	Rugby
Blue	Yes	Male	Short	Rugby
Brown	Yes	Male	Long	Rugby
Brown	No	Female	Long	Netball
Brown	No	Female	Long	Netball
Blue	No	Male	Long	Rugby
Brown	No	Female	Long	Netball
Brown	No	Male	Short	Rugby
Brown	Yes	Female	Short	Netball
Brown	No	Female	Long	Netball
Blue	No	Male	Long	Rugby
Blue	No	Male	Short	Rugby

Table 4.4.1

Table 4.4.1 shows an example of five attributes recorded for university students. If we apply the definition of Induced Dependencies we will have the following list of ID:

Sex→Class [1]
Class→Sex [1]
Eye color→Class [4/12]
Class→Eye color [5/12]
Eye color→ Sex [4/12]
Sex→ Eye color [5/12]

The above list of weak induced functional dependencies shows the interdependent attributes based on the underlying database shown in Table 4.4.1. The rest of the attributes have no interdependencies between one another and thus have no predictive power. In other words, it is logical to say the *class sex* and we mean that given the class we can determine the sex of the students and this can be written in the predicate form as *Student (class, sex)*. But what is the meaning of saying the *Married Hair length*? This is meaningless just because the underlying data in our example does not support any interdependency between the Married attribute and the Hair length attribute. As a result, the predicate *Student (married, Hair length)* represents no logic and having it as a part of a pattern will result in a pattern with no predictive value. The following is an example of a meaningless pattern:

Student (married, hair length)∧student (married, lives in)→student (hair length, lives in)

The user should be able to interfere to select the level of strength of the dependencies to be selected. This can be achieved by specifying a user-defined threshold for the selection of dependencies that have their strengths over the user-defined threshold. For instance, the married – hair length has dependency strength equal to 0 and it will be selected only if the user specifies his threshold to 0.

Generation of composite induced dependencies is also possible and has a major use in the generation of non-transitive patterns like pseudo transitive patterns that are discussed in section 4.8.1. From the above example we may infer the following composite dependency:

Eyecolor, Married, Hairlength → class.

This can be controlled by defining the maximum number of attributes on the left-hand side of the dependency. So by defining the maximum attribute number to two the above dependency will not be generated

4.5 Deducing overlaps from functional dependencies.

Functional dependencies can be used to deduce Foreign dependencies or overlaps. In other words, some of the overlaps can be deduced, rather than computed. Given two attributes C_x, and C_y then the overlap between the two attributes is defined as follows [30]:

Overlap (C_x, C_y) = max (|V_x ? V_y| / | V_x |, |V_x ? V_y| / | V_y |),
where V_x and V_y, are the value sets of C_x, and C_y respectively.

Since the computation requires an equi-join between the two over the values V_x and V_y. To avoid such joins, in some cases, we can make use of Domain Knowledge and functional dependencies. To illustrate we give the following example.

Let OL be the user defined threshold and given the relation R (A, B, C, D, E) with the following dependencies
A › B, C
AD › E

When normalizing, R will be decomposed into R1 (A, B, C) and R2 (A, D, E). Since one of the integrity requires for any value to exist in R2.A it should first exist in R1.A we can say the following:

ValueSet (R2.A)? ValueSet (R1.A)? ValueSet (R2.A)? ValueSet (R1.A) =ValueSet (R2.A) (1)

Let C_x = R1.A and C_y = R2.A and let V_x = ValueSet (R1.A) and V_y = ValueSet (R2.A).
Substituting the values in 1 we obtain that V_y ? V_x ? V_y ? V_x = V_y. Then in this particular case where V_y ? V_x = V_y, we can substitute the deduced value in the overlap equation. Thus, the overlap between two columns C_x and C_y can now be computed as:

Overlap (C_x, C_y) = max (|V_x ? V_y| / | V_x |, |V_x ? V_y| / | V_y |)
= Max (|V_y| / | V_x |, |V_y| / | V_y |)
= Max (|V_y| / | V_x |, 1)
= 1 since (|V_y| / | V_x |) ? 1

Since Overlap (C_x, C_y)=1 and 0 ? OL ? 1 then Overlap (C_x, C_y) ? OL, this means that in the case of foreign keys the overlap is always equal to 1, which is the maximum value an overlap can have.

Similarly given R (A, B, C) and the functional dependencies
A › B and B › C,
R decomposes into R1 (A, B) and R2 (B, C).
It is observed that ValueSet (R1.B) ? ValueSet (R2.B)? Overlap (R1.B, R2.B) = 1.

Based on the fact that only overlaps that are above the user defined threshold are added to the Significant Connection Table, any two attributes having similar interdependencies, as mentioned above, can be added to SCT. According to this result, we can make use of the dependencies both functional and induced to overcome the expensive computation of the overlap, which is based on a join between two attributes that have very large instances.

4.6 Computation of a pattern Strength

The final step in the generation of patterns is computing the strength of each against the underlying database. This step is very essential in the selection of interesting patterns, yet it is very expensive. The compute-strength (defined in section 3.3), indicates the probability of seeing the right hand side of a pattern p being true when the left-hand side of p is true, can be achieved by joining the tables over the overlapping attributes. As is known, the cost of joining tables is very expensive especially with large data volumes and considerable number of tables. To reduce this cost, we developed a new algorithm to find the number of true values on each of the two sides.

To compute the base and strength of each pattern we have to compute first $|S_{LHS}|$, where S_{LHS} is the set of tuples in U_{db} that satisfy LHS. The traditional way of computing S_{LHS} is by joining the tables that appear on LHS of the pattern and then counting the resulting set of records. Although this can be done by one select statement, its computational complexity is exponential.

We are not interested at this stage in computing the strength by the resulting set of records, but rather interested in the count of the resulting set instead. Consequently, we have derived a new algorithm to compute $|S_{LHS}|$ such that the computational complexity is reduced.
Given the pattern $T_1 (X_1, Y_1) \wedge T_2 (X_2, Y_2) \wedge \ldots \wedge T_n (X_n, Y_n) \rightarrow T_m (X_m, Y_m)$ and Y_i overlaps X_{i+1}, then to compute the strength and confidence of the LHS the only way is to make a join between tables $T_1 \ldots T_n$, over the overlapping attributes which as mentioned above very expensive, in terms of memory and time. Let also T_i have p_i tuples then the time complexity of these joins is $O(p_1 * p_2 \ldots * p_n)$. Assume $p = p_1 = p_2 = \ldots = p_n$, then it can be done in $O(p^n)$, where n is the length of the pattern.

The new algorithm will find the number of distinct occurrences of values of the column $T_1.Y_1$ and put the resulting tuples in temporary table or view, which we call Tcnt1. The next step is to do a join between Tcnt1 and T_2 over Tcnt1.$Y_1$and $T_2.X_2$ and compute the resulting distinct occurrences of $T_2.Y_2$. Then overwriting Tcnt1 by the resulting tuples. This action is done iteratively until we reach T_{n-1}. On T_n we do the same operation we did on T_1 and put the resulting set of records in Tcntn. The final operation we do for computing the cardinality of S_{LHS} is by computing the sum of the product from Tcnt1 and Tcntn.

The main enhancement in this algorithm is that we are always joining a relatively small table with a very large table. In other words, the number of tuples in Tcnt1 is always less or equal to the number of values in the domain of the Y_i attribute. The cardinality of S_{LHS} can thus be computed by the following algorithm:

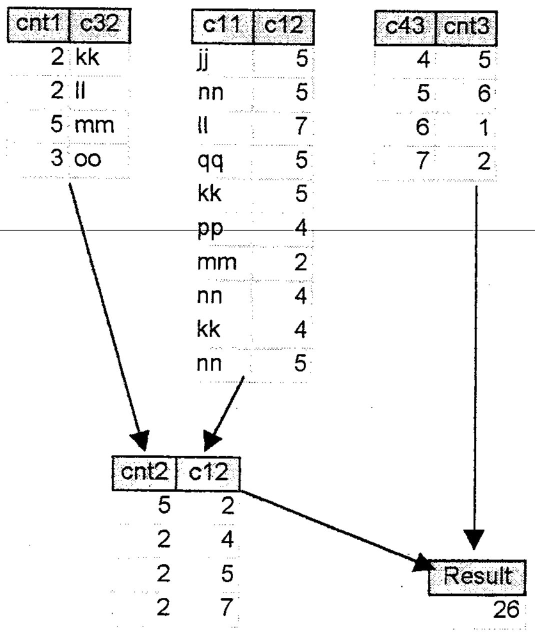
```
Function S_lhs ($T_1$ ($X_1$, $Y_1$)^$T_2$ ($X_2$, $Y_2$)^...^$T_n$ ($X_n$, $Y_n$)): integer
    Tcnt1, Tcntn, Tmp = table

    Begin
        If n = 1 then
                S_lhs = Select Count (*) from $T_1$;
        Else
                Tcnt1 = Select $Y_1$, Count (*) as Cnt1 From $T_1$ Group by $Y_1$
                Tcntn = Select $X_n$, Count (*) as Cntn From $T_n$ Group by $X_n$
                For i =2 to n-1 do
                        Tmp= Select $T_i$.Yi, Sum (Tcnt1.Cnt1) as Cnt1
                             From Tcnt1, $T_i$
                             Where Tcnt1.$Y_1$= $T_i$ $X_i$
                             Group by $T_i$.$Y_i$
                        Tcnt1=Tmp
                End for
                S_lhs = select sum (Cnt1*Cntn) from Tcnt1, Tcntn where Tcnt1.$Y_1$= Tcntn.$X_n$
        End if
    End
```

Fig. 3.5.1 Computing $|S_{LHS}|$ algorithm

For example, let us take the following transitive pattern generated from the database example in Figure 3.5.3. We will use the database example to illustrate the computation of the cardinality of the left-hand side.

T_3 (C_{31}, C_{32})$\wedge T_1$ (C_{11}, C_{12})$\wedge T_4$ (C_{43}, C_{43})$\rightarrow T_2$ (C_{21}, C_{23})



cnt1	c32
2	kk
2	ll
5	mm
3	oo

c11	c12
jj	5
nn	5
ll	7
qq	5
kk	5
pp	4
mm	2
nn	4
kk	4
nn	5

c43	cnt3
4	5
5	6
6	1
7	2

cnt2	c12
5	2
2	4
2	5
2	7

Result
26

Figure 4.6.1 An example for computing $|S_{LHS}|$

4.6.1 The computational complexity

For computing the time complexity, we proceed as follows: Let the table T_i have p_i tuples and the view Tcntn be defined as Tcntn = Select X_i, Count (*) as Cntr from T_i group by X_i

Also let the number of tuples in Tcntr = r, where r<<p and the number of tuples in Tcnt1 = q where q <<p.

Since the select statements contain a *Group By*, the data selected is sorted implicitly. Thus the complexity of the first select statement is O $(p_1 \log p_1)$ and the second is O $(p_n \log p_n)$. The cost of the select statement within the loop is O $(q_i p_i \log q_i p_i)$. For simplicity, assume that $p = p_1 = p_2 = \ldots = p_n$ and $q = q_1 = q_2 = \ldots = q_n$ then the computational cost of the entire loop is O (nqp log qp). Also, the cost of the last select statement is O $(r_i q_n)$. As a result, the time complexity of the whole algorithm is as follows:

$O(p_1 \log p_1) + O(p_n \log p_n) + O(nqp \log qp) + O(r_i q_n) = O(nqp \log qp)$

In the worst case where q = p, the time complexity is $O(np^2 \log p^2) << O(p^n)$. (The original time complexity of computing $|S_{LHS}|$).

This means that the time complexity for computing S_lhs this way is much less than building joins between the tables and then computing the number of tuples in the resulting table or view, where also the space allocated for the resulting vie is very large.

4.7 *Incremental Updating of discovered patterns*

Since it is costly to find the patterns in large databases, incremental updating techniques should be developed for maintenance of the discovered patterns to avoid redoing data mining on the whole updated database.

A database may allow frequent or occasional updates and such updates may not only invalidate some existing strong patterns but also turn some weak patterns into strong ones. Thus it is non-trivial to maintain such discovered rules or patterns in large databases. In our approach, incremental updating techniques are developed for efficient maintenance of discovered rules in databases with data insertion. The major idea is to reuse the computed overlaps and strengths of each pattern. Our new techniques are described in the following sections.

4.7.1 Optimization of computing the Overlap for incremental metapattern

Given two attributes C_x and C_y, the overlap between the two attributes is given by:

Overlap (C_x, C_y) = max ($|V_x ? V_y| / |V_x|$, $|V_x ? V_y| / |V_y|$)[30],
where V_x and V_y, are the values of C_x and C_y respectively.
Let V'_x be the values added to C_x.
Let V'_y be the values added to C_y.
New Overlap (C_x, C_y) =

Max ($|(V_x ? V'_x) ? (V_y ? V'_y)| / |(V_x ? V'_x)|$, $|(V_x ? V'_x) ? (V_y ? V'_y)| / |(V_x ? V'_x)|$).

Computing the overlap for every set of updates is very expensive since V_x and V_y are very large sets of tuples. By using the distributive law:

$(V_x ? V'_x) ? (V_y ? V'_y) = (V_x ? V_y) ? (V_x ? V'_y) ? (V_y ? V'_x) ? (V'_x ? V'_y)$, and instead of computing $(V_x ? V_y)$ after each set of updates, we will only compute $(V_x ? V'_y)$, $(V_y ? V'_x)$, and $(V'_x ? V'_y)$. The first two intersections are between a large set and a small set. The last one is an intersection between two small sets. The main improvement done is not to compute $(V_x ? V_y)$ except once.

4.7.2 Selection of overlaps to be computed after each set of updates

Let OL be the user defined threshold. $0 ? OL ? 1$.
As mentioned above, the computation of the overlap is expensive so as not to compute this value after each set of updates, where the added tuples will not raise the originally computed value above the user defined threshold, we will compute the Upper Limit of the overlap and compare it to the user-defined threshold. If the upper limit is greater or equal to the user-defined threshold, then we consider this overlap as promising and we then compute the overlap for only the promising overlaps. Since the computation of the upper limit is less expensive than the computation of the overlap,

Let $V1 = (V_x ? V_y) ? (V_x ? V'_y) ? (V_y ? V'_x) ? (V'_x ? V'_y)$

$V2 = (V_x ? V'_x)$

$V3 = (V_y ? V'_y)$

New Overlap (C_x, C_y) = Max ($|V1| / |V2|$, $|V1| / |V3|$)

Since the computation of |V1| is expensive, we compute its upper limit as follows:
Since the sets of records of V1 are mutually disjoined then:

$|V1|=|(V_x ?\ V_y)| + |(V_x ?\ V'_y)| + |(V_y ?\ V'_x)| + |(V'_x ?\ V'_y)|$

Since $|A| ?\ |A ?\ B|$ it follows that

$|V1| ?\ |(V_x ?\ V_y)| + |V'_y| + |V'_x| + \min(|V'_x|, |V'_y|)$,
which gives the upper limit of the new overlap. Using the upper limit of |V1| compare with the user defined threshold OL if greater or equal then compute the New overlap.

4.7.3 The Bayesian Approach

First update the dependencies, including foreign dependencies or overlaps, to see if a given pattern still holds. If it holds, then use the Bayesian formula to compute the strength of the instantiated patterns. The degree of a belief α is defined as a conditional probability, $P(\alpha|\xi)$, that α holds, given some previous evidence ξ supporting that belief. Given new evidence E, we update the degree of belief in α using a Bayes rule[33]:

$P(\alpha|E,\xi) = (P(E|\alpha,\xi)\,P(\alpha|\xi)) / (P(E|\alpha,\xi)\,P(\alpha|\xi) + P(E|\neg\alpha,\xi)\,P(\neg\alpha|\xi))$ Equation 3

Equation 3 assumes that for any belief α, $d(\neg\alpha|\xi)+d(\alpha|\xi)=1$. However one of the problems with the Bayesian approach is that it is sometimes difficult to compute the posterior probability $P(\alpha|E,\xi)$ from its prior probability $P(\alpha|\xi)$ because of the problems with computing the conditional probabilities $P(E|\alpha,\xi)$ and $P(E|\neg\alpha,\xi)$ used in the Bayes formula. To determine these probabilities, one has to assign, for each belief α, the joint probability distribution $P_\alpha(E,\xi)$ over the space of all possible pairs of (ξ, E), and this can be a formidable task.

To address this problem, these probabilities can be estimated using the following technique. Let α_0 be the belief that "the overall instructor ratings are higher than the overall course ratings" and assume that the degree of this belief is 0.85 ($P(\alpha_0|\xi) = 0.85$) and ($P(\neg\alpha_0|\xi)=0.15$). Assume that a new course evaluation arrived containing new evidence E_0 that the overall instructor rating for the course taught by this instructor is 5.8 and the over all course rating is 5.2. We associate two functions with belief α_0 computing the conditional probabilities $P(E|\alpha_0,\xi)$ and $P(E|\neg\alpha_0,\xi)$ for evidence E. Assume that it turns out that for the new evidence E_0, $P(E_0|\alpha_0,\xi)=0.62$ and $P(E_0|\neg\alpha_0,\xi)= 0.42$. Then substituting these numbers into Equation 3 we compute $P(\alpha_0|E_0,\xi)=0.89$

4.8 Generating Dependency Aided Patterns

The metapatterns are generated by finding cycles of overlapping attributes. The attributes are from different tables. The weakness in the original algorithm is that it focuses only on finding overlaps between attributes from different tables and then searching for cycles to generate the transitive metapatterns. This algorithm will generate only transitive metapatterns and does not make use of domain knowledge or Functional Dependencies or any kind of other dependencies.

By introducing dependencies as one kind of domain knowledge, the algorithm will be able to generate new form of rules that the original algorithm was not able to generate. The dependencies will be used to guide the algorithm in the generation process. The dependencies that can be introduced are Functional Dependencies, Induced Dependencies, Value Sensitive

Induced Dependencies, Weak Induced Dependencies[34], Joined Dependencies[35], and Fuzzy Functional Dependencies[36].

All the dependencies considered in this work except Functional Dependencies, are instance-based dependencies. In addition, for the important role that it plays in the database design it can play a very important role in the Knowledge discovery process. Functional Dependencies can be introduced in the discovery process to help in deducing overlaps, finding new non-transitive patterns, and predefine the dependencies that are presented in the form of predicates.

The Design-by-Example system of Kantola et al.,[37],[38] allows the building of E-R diagrams from an existing body of data. They show that for a naive dependency inference algorithm the time required is O ($n^2 2^n p \log p$) to find all dependencies in a relation with n attributes and p tuples. The mentioned algorithm is a naive one and there is plenty of room for improving its performance.

Although the time complexity of finding all functional dependencies is an exponential function, the exponential factor, which is the number of attributes, can be reduced in data- mining applications. Not all attributes are important in the data-mining process and some can be eliminated.

One means of avoiding problems associated with important attributes not being present in the training set is the use of constructive induction to create new attributes automatically from existing ones for subsequent use in the induction process. There are two principle approaches, Hypothesis-driven Constructive induction, and Data-driven Constructive Induction.[39]

Integrating the metapattern generator with human guidance will also help in reducing its complexity. Such users are the database designers and other expert users who can set up several functional dependencies and even reverse engineer a database design based on external knowledge.

By removing unimportant attributes, we are not only minimizing the number of attributes that the dependency generator will run over but also minimizing the possibility of generating uninteresting rules. Unimportant attributes are likely to result in rules with no predictive value.

The algorithm can be optimized in this fashion where the induced dependencies left over from previous runs are still valid and should not aim to regenerate the rule from scratch. In other words, the algorithm can compare the added set of tuples to rules derived from a previous run to see if any of the rules is violated. If the rule represents a soft belief, then it will be dropped.

Also the common attributes of the same rule are eliminated from the set of attributes to be included in next run of the algorithm. The algorithm does not have to find all solutions. It can be designed in a way that allows a graceful degradation of performance, as less inductive data is available until, at its bottom limit, the model degenerates to a normal relational database system.

Moreover, the original algorithm can not generate transitivity patterns from only one table because it looks for overlapping attributes from different tables. We have proven in our approach that we can generate transitivity patterns from only one table, which is discussed in 4.8.3

Since dependencies can be also viewed as simple patterns they can also be used as patterns in decision making. In addition to their use in decision making, dependencies and patterns can play a vital role in information retrieval planning. If the induced dependencies and patterns hold, provided that the confidence is 100%, they can aid in query reformulation and semantic optimization. Usefulness is also obvious in reverse engineering of databases. Relational patterns in section 4.8.1 are a very good example of patterns the can be used in semantic query optimization.

4.8.1 Generation of Non-Transitive Pattern

The original algorithm looks for overlapping attributes from different tables and then searches for cycles out of which it generates transitive patterns. Interesting patterns are not necessarily transitive and our approach seeks to uncover interesting hidden patterns, whether transitive or not based on their confidence and support, which the original algorithm fails to generate. By making use of functional dependencies, Induced Dependencies, Foreign dependencies, and fuzzy functional dependencies, we develop the new techniques for generating patterns. In the following part, we discus the new type of patterns that we generated which are Pseudo transitive, Additive, Extended transitive and Relational.

The original algorithm has several drawbacks even when generating transitivity patterns. It uses no criteria except the overlap for the selection of the attributes. As a result, many attributes could be selected that will result in patterns that have no predictive value. We can overcome this weakness by selecting attributes using the interdependency measure.

As mentioned in Attribute Selection section only attributes that are functionally interdependent are selected. And for consistency with Functional interdependency, Foreign interdependency is used in our approach instead of overlap. The following is the definition of the two different kinds of interdependencies

DEFINITION- Foreign Interdependency. *Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A Foreign interdependency* $X \Leftrightarrow Y$ *between X and Y exists in r if:*
$X \rightarrow Y$ (k) and $Y \rightarrow X$ (k') are weak Foreign dependencies and k or k' is above the user-defined threshold

DEFINITION- Functional Interdependency. *Let r be a nonempty (base or view) relation defined on R and let X and Y be subsets of attributes from R. A functional interdependency* $X \leftrightarrow Y$ *between X and Y exists in r if:*
$X \rightarrow Y$ (k) and $Y \rightarrow X$ (k') are weak functional dependencies and k or k' is above the user-defined threshold

In our approach, these two kinds of interdependencies are used in the generation process of transitive and non-transitive patterns as well. Even the original algorithm failed to generate some important transitive patterns as will be revealed in the next section. As a result, we developed a new approach for generating patterns based on the interdependencies defined above. In our approach the overlapping attributes or those having a Foreign interdependency do not have to be only from two different tables. They can be from different views instead, which will help, as we will see in the generation of previously hidden patterns. The following is the new terminology for generating new kinds of patterns.

Let V_i be the views defined over the underlying database, and let X_i be the attribute names. Also we use the "$\Leftrightarrow$" to denote a Foreign interdependency, and the double sided arrow "$\leftrightarrow$" to denote a functional interdependency. We, therefore, define:

1) Transitivity:

If we have the following interdependencies

$V_1.X_1 \leftrightarrow V_1.X_2$,

$V_1.X_2 \Leftrightarrow V_2.X_2$,

$V_2.X_2 \leftrightarrow V_2.X_3$,

$V_2.X_3 \Leftrightarrow V_3.X_3$,

$V_3.X_3 \leftrightarrow V_3.X_1$,

$V_3.X_1 \Leftrightarrow V_1.X_1$,

then the following cycle can be generated

$V_1 (X_1, X_2)$, V2 (X_2, X_3), V3 (X_1, X_3).

From the above cycle are the following set of patterns can be generated:

$V_1 (X_1, X_2) \wedge V_2 (X_2, X_3) \rangle\ V_3 (X_1, X_3)$,

$V_2 (X_2, X_3) \wedge V_3 (X_1, X_3) \rangle\ V_1 (X_1, X_2)$,

$V_3 (X_1, X_3) \wedge V_2 (X_2, X_3) \rangle\ V_2 (X_2, X_3)$.

For example: Citizen (X, Y)? Official-Language (Y, Z)› Speaks (X, Z);

2) Pseudo Transitive:

If we have the following interdependencies and weak induced dependencies

$V_1.X_1 \leftrightarrow V_1.X_2$,

$V_1.X_2 \Leftrightarrow V_2.X_2$,

$V_2.X_2, V_2.X_3 \rightarrow V_2.X_4$,

$V_2.X_4 \Leftrightarrow V_3.X_4$,

$V_2.X_3 \Leftrightarrow V_3.X_3$,

$V_3.X_1, V_3.X_3 \rightarrow V_4.X_4$,

$V_3.X_1 \Leftrightarrow V_1.X_1$,

then the following pattern can be generated

$V_1 (X_1, X_2) \wedge V_2 (X_2X_3, X_4) \rangle\ V_3 (X_1X_3, X_4)$.

For example:

Emp (Education, Position)∧Emp (Position, Experience, Salary) → Emp (Education, Experience, Salary)

3) Additive

If we have the following interdependencies and weak induced dependencies

$V_1.X_1 \Leftrightarrow V_2.X_1$,

$V_2.X_1 \Leftrightarrow V_3.X_1$,

$V_3.X_1 \Leftrightarrow V_1.X_1$,

$V_1.X_1 \leftrightarrow V_1.X_2$,

$V_2.X_1 \leftrightarrow V_2.X_3$,

$V_1.X_2, V_2.X_3 \rightarrow V_3.X_4$,

$V_3.X_1 \leftrightarrow V_3.X_4$

then

$V_1 (X_1, X_2) \wedge V_2 (X_1, X_3) \rightarrow V_3 (X_1, X_4)$.

For example: Emp (Position, Education)∧Emp (Position, Experience) → Emp (Position, Salary)

4) Extended Transitivity:

If we have the following transitivity pattern and the weak functional dependency

1- $V_1 (X_1, X_2) \wedge V_2 (X_2, X_3)$› $V_3 (X_1, X_3)$

2- $V_3.X_3 \rightarrow V_3.X_4$,

then

$V_1 (X_1, X_2) \wedge V_2 (X_2, X_3)$› $V_3 (X_1, X_4)$.

5) Relational

If we have the following set of interdependencies and weak dependencies

$V_1.X_1 \Leftrightarrow V_2.X_1$,

$V_2.X_2 \Leftrightarrow V_3.X_2$,

$V_1.X_1 \rightarrow V_1.X_3$,

$V_3.X_2 \rightarrow V_3.X_4$,

and $V_1.X_3$ and $V_3.X_4$ are defined on the same domain,

then we compare values of $V_1.X_3$, and $V_3.X_4$ and we select the relational operator that generates the highest number of tuples. Put the determining predicates on LHS and the resulting ones with the relational operator on the RHS to generate a pattern of the following format

$V_1 (X_1, X_3) \wedge V_2 (X_1, X_2) \wedge V_3 (X_2, X_4) \rightarrow V_1 (X_3)\ RO\ V_3 (X_4)$

where *RO* is a Relational Operator.

The compare function that compares the values of the attributes and returns the operator that has the highest number of tuples is defined as follows:

```
Function compare ($X_3$, $X_4$): relational operator
    Max=0
    For all relational operators
        Op = operator
        Select count (*) from $V_1$, $V_2$, $V_3$
        Where $V_1.X_1 = V_2.X_1$ and $V_2.X_2 = V_3.X_2$
        And $V_1.X_3$ Op $V_3.X_4$

        If count (*)>Max then
            Compare = Op
            Max = count(*)
        End If
    End For
End.
```

4.8.2 Experimental results

Emp ID	Education	Field	Position	Experience	Salary
1	Bac II	Math	Secretary	1	500
2	Bac II	Exp	Cash Reg	2	600
3	Bac II	Math	Cash Reg	2	600
4	Bac II	Phy	Show Rm	3	650
5	BS	CompSc	Programmer	1	750
6	BS	CompSc	Programmer	2	850
7	BA	Business	Sales	3	1000
8	MS	CompSc	Analyst	2	1500
9	MBA	Business	SalesMngr	1	1250
10	MBA	Intr Affairs	Comunic	1	1000
11	MS	CompSc	Proj Mngr	4	2000
12	PhD	CompSc	GM	10	3500
13	BacII	Phy	Sales	7	700

Table 4.8.1 An Employee Database Example

Pseudo Transitive

From the example database in Table 4.8.1 the *UpdateDependencies* function can find the following weak induced functional dependencies:

Position→Education [11/13] WFD-1
Position, Experience→ Salary [13/13] WFD-2
Education, Experience→ Salary [11/13] WFD-3

If the weak dependency threshold is less or equal to 11/13 then the above weak induced dependencies will be accepted. After updating SDT and CRT tables, the *Update (SCT)* can implicitly generate views to compute foreign dependencies or overalps. The views V_1, V_2, V_3 correspond to the weak dependencies WFD-1, WFD-2, WFD-3 respectively.

V_1

Education	Position
Bac II	Secretary
Bac II	Cash Reg
Bac II	Cash Reg
Bac II	Show Rm
BS	Programmer
BS	Programmer
BA	Sales
MS	Analyst
MBA	SalesMngr
MBA	Comunic
MS	Proj Mngr
PhD	GM
BacII	Sales

V_2

Position	Experience	Salary
Secretary	1	500
Cash Reg	2	600
Cash Reg	2	600
Show Rm	3	650
Programmer	1	750
Programmer	2	850
Sales	3	1000
Analyst	2	1500
SalesMngr	1	1250
Comunic	1	1000
Proj Mngr	4	2000
GM	10	3500
Sales	7	700

V_3

Education	Experience	Salary
Bac II	1	500
Bac II	2	600
Bac II	2	600
Bac II	3	650
BS	1	750
BS	2	850
BA	3	1000
MS	2	1500
MBA	1	1250
MBA	1	1000
MS	4	2000
PhD	10	3500
BacII	7	700

From the views V_1, V_2, V_3 we have the following weak induced Foreign dependencies:
V_1.Position $\rightarrow V_2$.Position [1]
V_2.Experience$\rightarrow$ V_3.Experience [1]
V_2.Salary$\rightarrow$ V_3.Salary [1]
V_3.Education$\rightarrow$ V_1.Education [1]

From the above set weak functional and foreign dependencies we will deduce the following set of functional and foreign interdependencies. These interdependencies are inserted to the new SCT that can keep information about interdependencies as well as on dependencies.

V_1.Education$\leftrightarrow$ V_1.Position
V_1.Position$\Leftrightarrow$ V_2.Position
V_2.Position, Experience$\rightarrow$ V_2.Salary
V_2.Experience$\Leftrightarrow$ V_3.Experience
V_2.Salary$\Leftrightarrow$ V_3.Salary
V_3.Education, Experience$\rightarrow$ V_3.Salary
V_3.Education$\Leftrightarrow$ V_1.Education

Based on the definition of pseudo transitive Patterns, from the above set of interdependencies the *Generate PseudoTransitive Metapatterns(DP)* will generate the following metapattern:
$P(x, y) \wedge Q(y, z, w) \rightarrow R(x, z, w)$ and finally the *GeneratePatterns(...)* will generate the following pseudo transitive patterns:

V_1 (Education, Position)$\wedge V_2$ (Position, Experience, Salary) $\rightarrow$ V_3 (Education, Experience, Salary)
$P_s=17/19$, $P_b=19/169$

V_1 (Education, Position)$\wedge V_3$ (Education, Experience, Salary) $\rightarrow$ V_2 (Position, Experience, Salary)
$P_s=21/31$, $P_b=31/169$

Additive

From the example database in Table 4.8.1 the *Update Dependencies (D, S)* will find the following weak induced functional dependencies:

Position→Education [11/13] WFD-1
Position→Experience [9/13] WFD-2
Education, Experience→ Salary [11/13] WFD-3

If the weak dependency threshold is less or equal to 9/13 then the above weak induced dependencies will be accepted. After updating SDT and CRT tables, the *Update (SCT)* can implicitly generate views to compute foreign dependencies or overalps. The vertical views V_1, V_2, V_3 correspond to the weak dependencies WFD-1, WFD-2, WFD-3 respectively.

V_1

Position	Education
Secretary	Bac II
Cash Reg	Bac II
Cash Reg	Bac II
Show Rm	Bac II
Programmer	BS
Programmer	BS
Sales	BA
Analyst	MS
SalesMngr	MBA
Comunic	MBA
Proj Mngr	MS
GM	PhD
Sales	BacII

V_2

Position	Experience
Secretary	1
Cash Reg	2
Cash Reg	2
Show Rm	3
Programmer	1
Programmer	2
Sales	3
Analyst	2
SalesMngr	1
Comunic	1
Proj Mngr	4
GM	10
Sales	7

V_3

Position	Salary
Secretary	500
Cash Reg	600
Cash Reg	600
Show Rm	650
Programmer	750
Programmer	850
Sales	1000
Analyst	1500
SalesMngr	1250
Comunic	1000
Proj Mngr	2000
GM	3500
Sales	700

From the views V_1, V_2, V_3 we have the following Weak Induced Foreign Dependencies:

V_1.Position →V_2.Position [1]
V_2.Position →V_3.Position [1]
V_3.Position →V_4.Position [1]

From the above set weak functional and foreign dependencies we will deduce the following set of functional and foreign interdependencies.

V_1.Position ⇔ V_2.Position
V_2.Position⇔ V_3.Position
V_3.Position⇔ V_1.Position
V_1.Position↔ V_1.Education
V_2.Position↔ V_2.Experience
V_1.Education, V_2.Experience → V_3.Salary
V_3.Position↔ V_3.Salary

Based on the definition of Additve Patterns, from the above set of interdependencies *Generate Additive Metapattern(DP)* and then *GeneratePatterns* will generate the following Additive pattern:

Emp (Position, Education)∧Emp (Position, Experience) → Emp (Position, Salary)
P_s=21/23; P_b=23/169

Relational Patterns

Ship

Name	Type	Length	Draft	Capacity
1	Tanker	500	4	500
2	Love	300	7	250
3	Cargo	400	5	350
4	Cargo	425	6	375
5	Fishing	250	10	150
6	Army	450	8	600

Port

Name	Country	Depth	Facilities
1	Lebanon	6	LNG
2	England	10	LNG
3	France	8	NOT
4	Dubai	9	LNG
5	Egypt	4	NOT

Visits

Ship	Port	Date	Qty
1	1	04/23/97	150
1	2	05/21/99	200
1	5	10/11/96	100
2	2	09/12/95	250
2	3	03/23/94	175
3	2	09/15/98	75
3	3	05/17/96	90
3	4	04/09/99	125
4	2	07/18/95	160
4	4	07/26/93	210
5	2	12/06/91	120
6	4	11/01/97	20

From the above database example the *UpdateDependencies* will generate the following weak functional dependencies

Ship. Name $\rightarrow$Ship. Draft [1]

Port. Name $\rightarrow$Port. Depth [1]

Based on deducing overlaps or Foreign Dependencies discussed in section 4.5 we can deduce the following Foreign Dependencies

Visits. Ship $\rightarrow$ Ship. Name [1]

Visits. Port $\rightarrow$ Port. Name [1]

Based on the interdependency definition and since the user-defined threshold can not be greater than one, we can have the above dependencies listed as the following interdependencies:

Ship. Name $\leftrightarrow$Ship. Draft

Port. Name $\leftrightarrow$Port. Depth

Visits. Ship $\Leftrightarrow$ Ship. Name

Visits. Port $\Leftrightarrow$ Port. Name

The next step in the generation of relational patterns is to have the Ship. Draft, and Port. Depth defined under the same domain. i.e. the two domains are comparable.
Since the attributes can be compared we will generate the following vertical view by joining the three tables over their overlapping attributes.

Draft	Depth
4	4
4	10
4	6
7	8
7	10
5	9
5	8
5	10
6	9
6	10
10	10
8	9

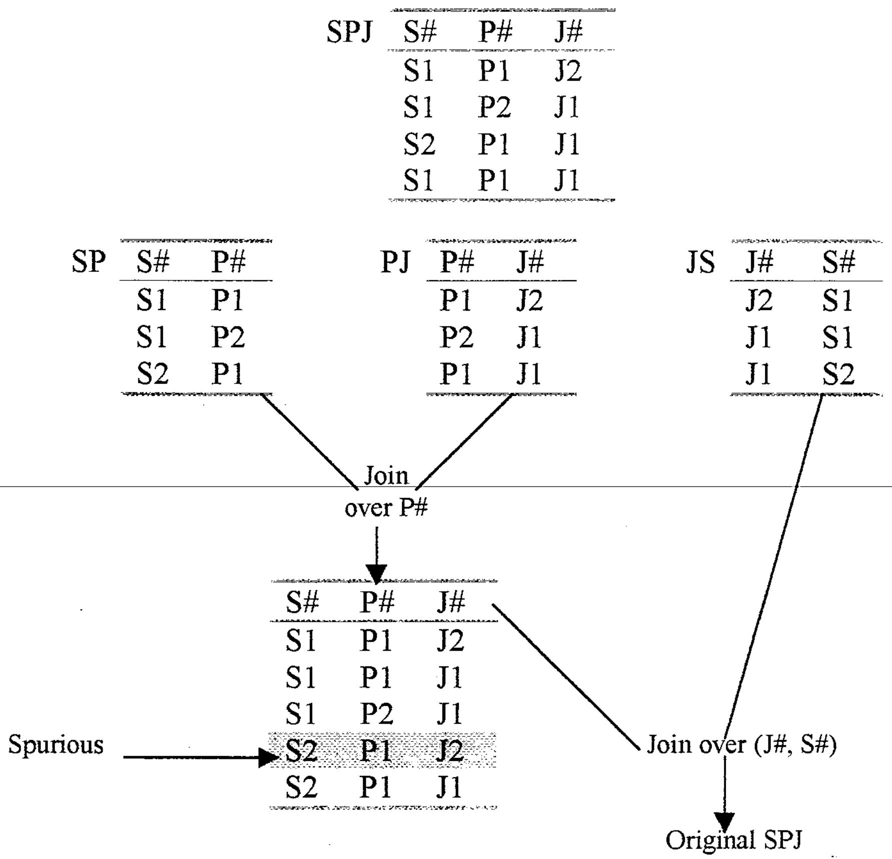
By applying the Compare function defined in 4.8.1 it will return the "$\leq$" relational operator.
Finally, by applying the last step in the generation process of relational patterns we put the determining predicates on the left-hand side and the resulting ones with the relational operator on the right hand side to get the following pattern:
Ship draft (X, Z)$\wedge$Visits (X, Y)$\wedge$Port-Depth (Y, W)$\rightarrow$Ship draft (X, Z)$\leq$ Port-Depth (Y, W)
$P_s=1$; $P_b=1/30$,
which can be also written as follows:
Ship (V) $\wedge$ Visit (V, P) $\wedge$ Port (P)$\rightarrow$Ship (draft)<Port (depth)
This reads as " a ship can visit a port only if the ships draft is less than or equal to the depth of the port." Such patterns can be used in semantic query optimization if they have strength equal to one, like the pattern generated above.

4.8.3 Join Dependency

The phenomenon of relations that cannot be losslessly decomposed into two projections but can be so decomposed into three or more and its association with transitive metapatterns is discussed in this section.

Consider relation SPJ [35]. The relation is "all key" and involves no nontrivial FDs or MVDs, and is therefore in 4NF. The figure below shows the three binary projections SP, PJ, JS of SPJ and the effect of joining SP and PJ over P# and then joining that result and JS over (J#, S#). The result of first join is a copy of the original SPJ relation plus one additional (spurious) tuple, and the effect of the second join is then to eliminate that tuple. In other words SPJ is equal to the join of its three projections SP, PJ and JS. This statement can be translated to the following condition:

If the pair (S1, P1) appears in SP,
and the pair (P1, J1) appears in PJ,
and the pair (J1, S1) appears in JS,
then the triple (S1, P1, J1) appears in SPJ (C1)



SPJ

S#	P#	J#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

SP

S#	P#
S1	P1
S1	P2
S2	P1

PJ

P#	J#
P1	J2
P2	J1
P1	J1

JS

J#	S#
J2	S1
J1	S1
J1	S2

S#	P#	J#
S1	P1	J2
S1	P1	J1
S1	P2	J1
S2	P1	J2
S2	P1	J1

Figure 4.8.1 SPJ is the join of all three of its (binary) projections but not of any two.[35]

Since (S1, P1) appears in SP if and only if (S1, P1, J2) appears in SPJ for some J2, similarly for (P1,J1) and (J1,S1), the above condition can be also written on SPJ as follows:

If (S1, P1, J2) and (S2, P1, J1) and (S1, P2, J1) appears in SPJ *then* (S1, P1, J1) appears in SPJ.(C2)

As defined in [40], *A relation will be n decomposable (for n>2) if and only if it satisfies a cyclic constraint.* The cyclic nature of the constraint that appears clearly in the conditions above, makes these conditions very much similar to transitivity patterns defined in [30]. The only difference between the conditions and the patterns is the confidence. In other words, conditions can be viewed as patterns with 100% confidence. Thus the above conditions can be written in a predicate format as follows:

$SP(X, Y) \wedge PJ(Y, Z) \wedge JS(Z, X) \rightarrow SPJ(X, Y, Z)$ $P_s=1$ and $P_b=4/27$ (P1)

P1 can be also written as a pattern over SPJ as follows:

$SPJ(X, Y, C) \wedge SPJ(A, Y, Z) \wedge SPJ(X, B, Z) \rightarrow SPJ(X, Y, Z)$ $P_s=1$ and $P_b=4/64$ (P2)

From the definition of Join dependency in [35], A relation R satisfies the join dependency (JD)

* (X, Y, …, Z) if and only R is equivalent to the join of its projections on X, Y, …, Z, where X, Y, …, Z are subsets of the set of attributes of R.

In the SPJ example, SPJ being the join of its projections SP, PJ, JS is consistent with condition C1. Also, if we have the condition C1 then SPJ can be decomposed into SP, PJ, and SJ. By applying the algorithm of the metapattern generation to SP, PJ, and SJ in Figure 4.8.1, we will find the cycle SP (X, Y),PJ (Y, Z),JS (X, Z) out of which the following patterns are generated:

$SP(X, Y) \wedge PJ(Y, Z) \rightarrow JS(X, Z)$ $P_s=4/5$ and $P_b=5/9$ (P3)

$PJ(Y, Z) \wedge JS(Z, X) \rightarrow SP(X, Y)$ $P_s=4/5$ and $P_b=5/9$ (P4)

$SP(Y, X) \wedge JS(X, Z) \rightarrow PJ(Y, Z)$ $P_s=4/5$ and $P_b=5/9$ (P5)

Notice that the strength of the above patterns is 4/5 and not 5/5 because the spurious record pointed at in Figure 4.8.1 is the only record in the LHS that does not satisfy the RHS.

From the above observation we can deduce that if all of the transitivity patterns generated from a cycle of the form SP (X, Y), PJ (Y, Z), JS (X, Z) have a confidence (strength) equal to one, then the relation SPJ satisfies the Join dependency JD *(SP, PJ, SJ). In addition to the use of the discovered transitivity patterns in decision making, the deduced join dependency can be used in reverse database engineering.

The original algorithm generates the transitivity patterns by searching for cycles of overlapping attributes from different tables. But we observe that we can induce transitivity patterns from only one table and even without searching for cycles of overlaps. As defined in [35], if condition C2, which is equivalent to the hard pattern P2, holds then the table SPJ is three decomposable. Applying the metapattern generator to the projections of SPJ, we will generate the three patterns P3, P4, P5 listed above. As a result, given the hard pattern P2 we can deduce the patterns P3, P4, and P5. The spurious records are the only factor that reduces the confidence of the deduced patterns. Nevertheless, the deduced patterns can still be interesting. The original algorithm will miss the generation of the patterns similar to P3, P4, and P5 if the database is not fully normalized to 5NF.

Moreover, we observed the possibility of generating transitivity patterns from only one table. We can generate transitivity patterns by searching for cycles of interdependent attributes of the same table. To make it clearer, we will explain our approach in the following example.

In section 4.4, we discussed attribute selection and attribute interdependencies. Taking the same database example as in Table 4.4.1 and the weak induced dependencies generated from that example. given as

(Sex→Class [1]
Class→Sex [1]
Eye color→Class [4/12]
Class→Eye color [5/12]
Eye color→ Sex [4/12]
Sex→ Eye color [5/12]),

and if the user-defined threshold is less than 5/12, we can then say that we have the following functional interdependencies:

Sex↔Class, Class↔Eye Color, and Eye Color↔Sex.

Observe the cyclic nature of the above interdependencies, which in our approach, is the requirement for generating transitivity patterns. As a result, we will generate the view SCE from Table 4.4.1 by selecting and grouping by Sex, Class, and Eye Color. Where the result is the following:

Sex	Class	Eyecolor
Female	Netball	Brown
Male	Rugby	Blue
Male	Rugby	Brown

We will also generate the views SC, CE, and ES, which correspond to the interdependencies respectively.

SC

Sex	Class
Female	Netball
Male	Rugby

CE

Class	Eyecolor
Netball	Brown
Rugby	Blue
Rugby	Brown

ES

EyeColor	Sex
Blue	Male
Brown	Female
Brown	Male

Since we have in addition to the functional interdependencies the following Foreign interdependencies or overlaps:

SC.class↔CE.class, CE.eyecolor↔ES.eyecolor, and ES.sex↔SC.sex.

Then we can have the following cycle: SC(sex, class), CE(class, Eyecolor), ES(Eyecolor, Sex), out of which we can generate the following patterns:

SC (X, Y)$\wedge$CE (Y, Z)→ES (X, Z) P_s=1 P_b=3/6,
CE (Y, Z)$\wedge$ES (Z, X)→SC (Y, X) P_s=3/5 P_b=5/9,
ES (Z, X)$\wedge$SC (X, Y)→CE (Z, Y) P_s=1 P_b=3/6.

Generation of transitivity patterns and other kinds of patterns from the same table or by having some of the Foreign Dependencies generated from the same table is a key issue for generation of patterns from denormalized databases. This is an important issue in Data Warehouses where the data is denormalized for performance issues. Now, we can generate interesting patterns as the ones generated in our approach with out having to have normalized databases as in the original algorithm.

4.9 Generating Horizontally weak Patterns

In pattern generation algorithms, the patterns are evaluated against the underlying database and a confidence measure is computed. The confidence is a belief measure of interestingness or strength of a pattern. Confidence weakening can be thought of as vertical weakening, where the confidence of a pattern will be affected by the augmentation of new predicates to the pattern. As a result, different patterns have different strengths. Patterns that have their confidence computed over the whole underlying database are called Full patterns. In previous algorithms, confidence weakening was applied only to Full patterns.

In our work, we note that Full patterns are not the only patterns that can be generated. Each pattern is supported by the tuples in the underlying database. Instead of having the whole underlying instances as a support for a given pattern, the case of Full patterns, a selection of the instances can do the job in supporting new kind of patterns. Since only part or a horizontal view is supporting the new kind of patterns and the selection will narrow or weaken the supporting set of data, we appropriately call them horizontally weak patterns.

The new kind of patterns made it possible to generate further previously non-generated patterns. The combination of confidence weakening and horizontal weakening will enable us to generate pattern which are best called Soft Value Sensitive patterns. The following figure will illustrate the idea.

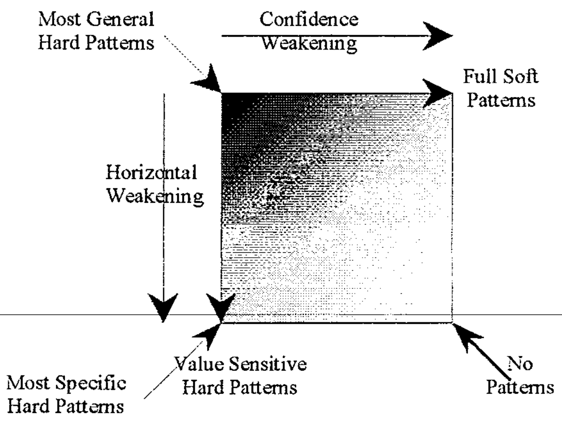

Most General Hard Patterns
Confidence Weakening
Full Soft Patterns
Horizontal Weakening
Value Sensitive Hard Patterns
Most Specific Hard Patterns
No Patterns

Fig. 3.5.1. Horizontal and Confidence weakening in Patterns.

In the metapattern generation algorithm, The degree of belief or (CompStrength) of a given pattern is represented by two parameters *Strength* and *Base*. While in other algorithms, like generation of association rules, the degree of a belief is measured by *Confidence* and *Support*. Confidence denotes the strength of implication and support indicates the frequencies of the occurring patterns in the rule. The degree of belief of a pattern can also be thought of as the confidence weakness of a pattern. Beliefs with high degrees are called Hard beliefs, and those with low degrees are called Soft beliefs. As a mater of fact, beliefs of a pattern range between the Hard and the Soft.

Weakening of patterns can be observed in two dimensions. The first is confidence weakening and the second is horizontal weakening. Confidence weakening of a pattern weaken the pattern in confidence manner in that the target predicates only adhere to the pattern to a given confidence level. Horizontal weakening is that not all tuples support the pattern and a selection must be made to increase the pattern confidence. The selection can be done by classification or splitting domains into sub domains. Patterns range from Hard (*strong*) to the soft (*weak*) and from the Full (*The Most General*) to the (*the most specific*). The hard or strong patterns are those with a confidence or belief equal to 1. The soft or weak patterns are patterns that do not have enough support and having a confidence nearly 0. Full patterns or the most general patterns are those whose confidence or support is computed over the whole domain or over all classes. The most specific patterns are those where their confidence is computed over a class or set of tuples.

Unlike previous algorithms, that generated only the most general patterns, the New algorithm is capable of generating Patterns that range from the most general to the most specific and from the most soft to the most hard.

4.9.1 Generating Value Sensitive patterns

A value sensitive pattern is a pattern having one or more of its predicate parameters as a reference to a sub-domain of a given attribute domain. Some of the patterns do not hold over the whole underlying domain, while they do hold in sub-domains. As a result, a selection should be made as to increase the possibility of having overlaps above the user-defined threshold. Generating value sensitive patterns is very important since these kinds of patterns are more interesting to users or decision-makers than full patterns. Generating previously unknown interesting patterns, the key issue in data mining, is the objective behind generating value sensitive patterns. Previous algorithms did not generate such patterns.

In the new algorithm, generating value sensitive patterns starts at the level of computing overlaps. If an overlap between two attributes of the same type over the whole domain is below the user-defined threshold, the new algorithm will look for overlaps over sub-domains of the same attributes. Once an overlap of this nature is found an entry for the sub domains is added to the Ranges Table and a reference name is added to the Significant Connection Table, for finding cycles later on. After the patterns are generated, that reference name will appear as a predicate parameter, but instead of referencing an attribute name it will reference the sub-domain of an attribute.

DEFINITION: ***Value Sensitive Pattern***. Let *p* be a pattern and R the set of attributes in *p*. Let r be a nonempty (base or view) relation defined on R . Let X and Y be two attributes of the same type and subset of R. Let D_x and D_y be the domains of X and Y, respectively, and let D'_x be a proper subset of D_x and D'_y be a proper subset of D_y. A value sensitive pattern exists with respect to D'_x and D'_y if Overlap (X', Y') holds in r' where $r' = \sigma_{x \in D'_x \wedge y \in D'_y}$ (r) and r' is nonempty.

Three points should be taken into account.

1) When D'_x is equal to D_x and D'_y is equal to D_y, then the Value Sensitive Pattern simple becomes a Full Pattern, as the select operations specify no restriction.
2) Restricting the domain of X does not require a corresponding restriction in the domain of Y and vice versa.

3) A qualitative measure of the weakening of the Full Pattern is the extent to which D'_x and D'_y are reduced relative to D_x and D_y.

Selection of sub-domains is very essential in the generation of value sensitive patterns. The selected sub-domains affect interestingness of patterns. As a result, the selection of subdomains must be based on the following conditions or constraints:

1) Have an overlap above the user-defined overlap threshold.
2) Represent a meaningful class.
3) Having a card(sub-domain)/card(domain) above a user defined threshold
4) Having the ratio of values in sub-domain to values in the whole domain above a user defined threshold.

Notice that not all overlaps result in interesting patterns. We say an overlap is "promising" only if its significance is above some user specified thresholds. In the new approach, each overlap is evaluated by two horizontal values: the sub-domain ratio S_r which reflects how much the sub-domain covers of the whole domain, and the sub-domain base S_b which reflects how much the sub-domain is supported by the actual data in the database.

The ratio value is computed as $S_r = | \text{SD} | / | \text{D} |$, where SD is the set of distinct values in the sub-domain and D is the set of values in the Domain.

The base value is computed as $S_b = |\text{Values (SD)}| / |\text{Values (D)}|$, where Values (SD) is the set of tuples in the database for which the sub-domain is true, and Values (D) is the set of tuples in the database of the whole domain.

A sub-domain's ratio and base values, S_r and S_b, are compared with two user specified thresholds T_r and T_b. When both the ratio and base are above their thresholds, the sub-domain is accepted.

Value Sensitive patterns range from the most general to the most specific. If the most general pattern can be generated, then it is worthless looking for its specific patterns. Only when a pattern does not have a confidence above the user-specified threshold, looking for specific patterns becomes feasible. To illustrate we will take the following example:

Author	Title	Subject	Type	Language
Orwell	Animal Farm	Literature	Novel	English
Shakespeare	Hamlet	Literature	Novel	English
Dostoivsky	Dr. Zivago	Literature	Novel	Russian
Turgenev	Fathers&Sons	Literature	Novel	Russian
Steven King	Pet Sematary	Literature	Novel	English
Daniel Steal	Star	Literature	Novel	English
Tawfik Awad	Alraghif	Literature	Novel	Arabic
Einstein	Relativity Theory	Science	Physics	English
Ibn Sina	Arabic Medicine	Science	Medicine	English
Al Gabr	Algebra	Math	Algebra	English
Al Khawarizmi	Algorithmics	Science	Comp Sc	English
Marie Curie	Atom theory	Science	Physics	English
Marx	Surplus Value	Social Science	Politics	English

Edison	Electricity	Science	Physics	English
Newton	Force & Gravity	Science	Physics	English

Writer	Writes in
Orwell	English
Shakespeare	English
Dostoivsky	Russian
Turgenev	Russian
Steven King	English
Daniel Steal	English
Tawfik Awad	Arabic
Einstein	German
Ibn Sina	Arabic
Al Gabr	Arabic
Al Khawarizmi	Arabic
Marie Curie	Polish
Marx	German
Edison	French
Newton	English

Figure 4.9.1 A library database example

Instantiation of a metapattern with table and attribute names is the level at which the metapatterns are transformed into patterns. For example, given the metapattern

$T_1 (X_1, X_2) \wedge T_2 (X_2, X_3) \rightarrow T_3 (X_1, X_3)$

Applying the algorithm of generating transitivity patterns on the above database example, one of the patterns that can be generated is the following:

AuthorOf (Author, Title) $\wedge$ writtenIn (Title, Language)$\rightarrow$ canWrite (Writer, Writes in)

P_s=8/15; P_b=15/225,

which can be also written as follows:

authorOf (x, y) $\wedge$ writtenIn (y, z) › canWrite (x, z)[0.53, 0.07]

The above pattern is the most general pattern. In other words, predicate parameters are instantiated by attribute names covering the whole domain. The most general pattern might not hold over the whole domain while it can hold on a sub-domain of one or more of the attributes. The pattern that holds in sub-domains is called a value sensitive pattern.

Since the above pattern has a low strength value, then we can search for a sub domain that will increase the confidence of the above pattern. To do so, we will search for this sub-domain in the attributes that are determined (to a certain extent) by the attributes appearing in the pattern. In our example, we have the following weak induced dependency:

Title$\rightarrow$Type [1]

```
SELECT AuthorOf.Type, count (*) as Count
FROM AuthorOf, writtenIn, canWrite
WHERE AuthorOf.author= canWrite.writer and writtenIn.title= AuthorOf.title
And canWrite writesin= writtenIn.language
GROUP BY AuthorOf.Type.type;
```

The above select statement will generate the following View:

Type	Count
Novel	7
Physics	1

The sub-domain is defined by selecting the items that have a count/whole number of records over a user-defined threshold.

Suppose the user defined threshold is 0.3 and the ratio novels is 7/15=0.46, then "Novel" is selected a one item sub domain

authorOf (x, y) ∧ writtenIn (y, z) ∧ Type (y, 'Novels') › canWrite (x, z), with
P_s=7/7; P_b=7/49

Now we have a value sensitive Pattern with a higher strength value. As a result, the above value sensitive pattern is output while the most general one is not.

As value sensitive patterns can range from the most general to the most specific, the above pattern can become more specific by instantiating other predicate parameters by reference to a subdomain or a class. For example:

authorOf (x, 'Novels') ∧ writtenIn ('Novels', 'Arabic') › canWrite (x, 'Arabic'),
or
authorOf (x, 'Horror Novels') ∧ writtenIn ('Horror Novels', z) › canWrite (x, z).

The above two patterns are more specific than the first because the number of tuples supporting it is less or equal to the number of tuples supporting the first one. The instantiation can be interpreted as constraints that will minimize the number of underlying tuples. The specificity of a pattern can go down to the most specific pattern where the support is just a one tuple. Instantiating the pattern by the atomic values will result in the most specific pattern. The following is an example of such patterns.

authorOf ('Awad', 'Alraghif') ∧ writtenIn ('Alraghif', 'Arabic') › canWrite ('Awad', 'Arabic').

In general the most specific patterns are rarely interesting, if ever. To avoid the generation of such patters, a new threshold for the percentage of the supporting tuples is assigned and only those patterns having a value above the user-defined thresholds are accepted.

4.10 The New Algorithm Put Together

SDT = Sub-Domain Table (SD-Id, Table name, attribute name, Domain/Sub-domain)
CRT = Cardinality Table (SD1, SD2, card (SD1 ? SD2)) SD1&SD2 are of the same Type
SCT = Significant Connection Table
RFN = Reference Name
FD = Functional Dependencies
ID = Induced Dependencies
VSID = Value Sensitive Induced Dependencies
WID = Weak Induced Dependencies
FFD = Fuzzy Functional Dependencies
JD = Joined Dependencies

```
Procedure DataMiner (D as DataBase, S as Schema, OT, b, s as threshold);
    D is the database instance
    S is the table schemas and integrity constraints
    OT the user defined overlap threshold
    b is the user defined base threshold
    s is the user defined strength threshold
    The Dataminer is the main procedure that calls other procedures
    It makes use of dependencies both functional and induced.
    It looks for overlaps in sub domains that are in Sub-Domains Table (SDT).
    SDT: is as defined above keeps entries of attribute sub-domains with overlaps.
    DP1 is the set of defined functional and Foreign dependencies
    DP2 is the set of induced functional and foreign dependencies.
    M is the set of MetaPatterns
    Begin
        DP1 = GetDependencies (D, S) //reads dependencies defined at design time
        For each set of Updates
                DP2=Update Dependencies (D, S)
                DP=DP1 U DP2
                Update CRT (D, SDT)
                Update SCT (CRT, DP, OT)
                Generate MetaPattern (SCT)
                Generate Patterns (M, D, DP, b, s)
        End for
    End.
```

```
Procedure Update Dependencies (D as DataBase, S as Schema, )
  $T_l$ is the user-defined threshold of the number of attributes appearing on the LHS of a
     dependency.
  $T_r$, $T_b$ are the ratio and base thresholds respectively of the value sensitive dependency
  $T_c$ is the confidence or strength threshold of the weak dependencies
  Each of the mentioned functions below will update the list of dependencies that reflect the
  under lying database after each set of updates. The functions will apply the definitions of the
  dependencies defined in previous sections.
  Begin
    Update (ID, $T_l$) // Updates Induced Dependencies with LHS length $\leq T_l$
    Update (VSID, $T_r$, $T_b$) // Updates Value Sensitive Induced Dependencies and outputs those
                          with ratio and base above $T_r$, $T_b$
    Update (WID, $T_c$) // Updates Weak Induced Dependencies and outputs those with
                        confidence above $T_c$
    Update (FFD) // Updates Fuzzy Functional Dependencies
    Update (JD) // Updates Joined Dependencies
    Update (SDT)// Updates this table with any new domains or sub-domains especially when
    new VSIDs are found
  End

Procedure Update CRT (D as DataBase, SDT as table)
  D is the database instance
  SDT: is as defined above keeps entries of attribute sub-domains with overlaps.
  Update CRT keeps an updated version of the intersection between any two sub-domains of the
  same type found in SDT. The purpose of this procedure is to keep the intersection between two
  sub-domains computed in previous runs and update it using the distributive law described in
  section 4.7.1
  Begin
    For each two sub-domains of the same type in SDT
      Find Card (SD1 ?  SD2)
      If record not found then
        Insert the record into CRT
      Else
        Update record in CRT
      End If
    End for
  End
```

```
Procedure Update SCT (CRT as table, DP as Dependencies, OT as Threshold)
  CRT remembers the cardinality of the intersection between two sub-domains
  DP is the set of dependencies
  OT is the user defined Overlap Threshold
  Update SCT updates the Significant Connection Table, which keeps entries of
  interdependencies and not only overlapping sub-domains.
  It deduces overlaps from dependencies.
  It computes the overlap only for promising sub-domains after each set of updates. See section
  4.7.2.
  If no overlaps (foreign dependencys) were found between the whole domain of two attributes it
  searches for overlaps in the sub-domains. If patterns were then found for these attributes,
  Value Sensitive Patterns are then generated.
  Begin
    For the changes in dependencies
      Deduce Overlaps using ID and add an RFN to SCT for each
      Deduce Overlaps using FFD and add an RFN to SCT for each
      Deduce Overlaps using JD and add an RFN to SCT for each
    End For

    For each two column sub-domains $C_x$ & $C_y$ in SDT from different views
      If $C_x$ & $C_y$ are of the same type then
        UL = Upper Limit ($C_x$, $C_y$). See section 4.7.2
        If UL>= OT then
          OL =Overlap ($C_x$, $C_y$)
          If OL >= OT and not in SCT then
            Create RFN ($C_x$, $C_y$) and add to SCT
          Else
            Search for new sub-domains where OL>= OT
            If any is found then
              1- Create a new sub-domain entry in the SDT
              2- Update CRT (D as DataBase, SDT as table)
              3- Create RFN ($C_x$, $C_y$) and add to SCT
           End if
        End if.
      End if
    End For
  End.
```

```
Procedure Generate MetaPattern (SCT as Table, DP as Dependencies)
    SCT is the Significant Connection Table. (Interdependency Table)
    DP is the set of dependencies.
    Generate the metapatterns with the help of interdependencies stored in SCT.
    Begin
        Generate Transitivity Metapattern(DP) // Searches for transitivity patterns as defined in
        4.8.1
        Generate PseudoTransitive Metapattern(DP) Searches for Pseudo transitive patterns as
        defined in 4.8.1
        Generate ExtendedTransitive Metapattern(DP) Searches for Extended transitive patterns as
        defined in 4.8.1
        Generate Additive Metapattern(DP) Searches for Additive patterns as defined in 4.8.1
        Generate Relational Metapattern(DP) Searches for Realtional patterns as defined in 4.8.1
    End

Procedure Generate Patterns (M as MetaPatterns, D as DataBase, DP as Dependencies, b, s as
Threshold)
    M is the set of generated Metapatterns.
    D is the database instance.
    DP is the set of dependencies.
    b is the user defined base threshold
    s is the user defined strength threshold
    $P_s = Prob(RHS|LHS, U_{db}, I_o) = (|S_{RHS}|+1) / (|S_{LHS}|+2)$
    $P_b = (|S_{LHS}|) / Dom(LHS)$
    $S_{LHS}$ is the set of tuples in $U_{db}$ that satisfy LHS.
    $S_{RHS}$ is the set of tuples in $S_{LHS}$ that satisfy RHS.
    Loop
        Order and Display M;
        Let User examine, create, and reorder M;
        Select m from M;
        For each Pattern p instantiated from m;
            Compute the strength value $P_s$ and the base value $P_b$; Use Bayesian formula
            If $P_s >= s$ or $P_s <= 1-s$ then
                Output (p);
            Else if $P_b >= b$ then
                Search for sub-domains in attributes determined (even weakly) by attributes
                appearing in p
                Create new metapattern by adding the constraint to the left-hand side of p;
                Insert the new mettapattern to M;
            End if
    Until M is empty or the user instructs to stop;
    End.
```

Conclusions

This work has presented an approach that enhanced both the performance and the quality of the generated metapatterns. The original metapattern generator is combined with dependencies, such as Functional Dependencies and Foreign Dependencies, to generate patterns other than transitivity patterns. Experimental results have shown that with this approach pseudo-transitive, additive, relational, and extended transitive patterns can be generated. This new form of patterns can now be generated by making use of domain knowledge and exploiting existing data dependencies.

Another significant contribution of this work is the notion of generating value sensitive patterns. Since patterns can be defined over sub-domains rather than over a whole domain, the new approach will make use of value sensitive dependencies and incorporate the idea of finding overlaps in sub-domains to generate value sensitive or class based patterns.

Furthermore, the algorithm in its present form is impractical and does not depart from the realms of theoretical presentation. Its impracticality is partly because it is incapable of updating previously generated metapatterns. As patterns generated should always reflect the underlying data, the patterns should be updated as the underlying data changes rather than regenerated. As a result, Bayes formula and patterns that were generated in previous runs are used in our approach to incrementally update patterns after each set of updates that take place in the database.

One of the drawbacks of the original algorithm is that the evaluation of the patterns against the underlying database mainly relies on carrying out joins between several tables. This threatens seriously the practical feasibility of the technique as the database increases in size. The new approach overcomes this draw back by using a new technique for evaluating the patterns against the underlying database.

In our new approach, domain knowledge is not only used to generate previously non-generated patterns, but also used to deduce overlaps. By deducing some of the overlapping attributes without computing the overlap we will improve the performance of the algorithm.

The selection of attributes is also a vital issue in inductive learning. In the traditional algorithm, the only criterion for selecting an attribute is to have an overlap with another attribute from a different table. This kind of selection of attributes can result in patterns with no predictive value. In our approach, for an attribute to be selected, it should be interdependent with another attribute in addition to the overlap. Moreover, in our approach an overlap between two attributes of the same table is possible. This notion enables us to generate patterns from denormalized databases that the original algorithm failed to generate.

References

[1]G. piatetsky-Shapiro and W.J Frawley, "Knowledge Discovery in Databases." AAAI/MIT Press, 1991.

[2]Usama M. Fayyad , "Data Mining and Knowledge Discovery: Making Sense Out of Data.", IEEE Expert, October 1996.

[3]Ming-Syan Chen, Jiawei Han, and Philip S. Yu, "Data Mining: An Overview from a Database Perspective", *IEEE Transaction on Knowledge and Data Engineering*, VOL. 8, NO. 6, Dec 1996.

[4] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, " From Data Mining to Knowledge Discovery: An Overview," *Advances in knowledge Discovery and Data Mining*, MIT Press, Cambridge, Mass, 1996, pp. 1-36

[5] Thomas G. Dietterich and R. Michalski, "A comparative review of selected methods for learning from examples." *Machine Learning, an Artificial Intelligence approach*, Vol. 1. Morgan Kaufmann, San Mateo, California, 1983

[6] Marcel Holsheimer, Arno Siebes, "Data Mining: The Search for Knowledge in Databases.' *Thesis Report, Computer Science Department*, Amesterdam, The Netherlands, 1994.

[7] Rakesh Agrawal, Tomaz Imielinski, and Arun Swami, "Mining Association rules between sets of items in large databases," *In Proc. Of the ACM SIGMOD Conference on Management of Data*," pp. 207-216, Washington, D.C., May 1993.

[8] Rakesh Agrawal, and John Shafer, "Parallel Mining of Association Rules," *IEEE Transactions on knowledge and Data Engineering*, Vol 8, No 6, pp. 962-969, December 1996.

[9] Ramakrishnan Sirkant, Rakesh Agrawal, "Mining Generalized Association Rules," Proceedings of the 21st VLDB Conference, Zurich, Switzerland, 1995.

[10] Kamal Ali, Stefanos Manganaris, and Ramakrishnan Srikant, " Partial Classification using Association Rules," *AAAI/IBM Almaden Research Center*, 1997.

[11] Rakesh Agrawal, Ramakrishnan Srikant, "Fast Algorithms for mining Association Rules," In *Proceedings of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September 1994.

[12] Ryszard S. Michalski, "A theory and methodology of inductive learning." *Machine Learning, an Artificial Intelligence approach*, Vol. 1. Morgan Kaufmann, San Mateo, California, 1983

[13] IBM Corporation, "Data Management Solutions," *IBM's Data Mining Technology*, Stamford, Connecticut, April 1996.

[14] J. Ross Quinlan, "Induction od Decision Trees," *Machine Learning*, pp. 81-106, 1986.

[15] L. Breiman et. al, "Classification and Regression Trees," Wadsworth, Belmont, 1984.

[16] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A Fast Scalable Classifier For Data Mining," *Proc. Int'l Conf. Extending Database Technology (EDBT' 96)*, Avignon, France, Mar 1996.

[17] Information Discovery, Inc. "Rules are Much More than Decision Trees," www.datamining.com, 1996

[18] Hongjun Lu, Rudy Setiono, and Huan liu, "Effective Data Mining Using Neural Networks," *IEEE Transaction on Knowledge and Data Engineering*, VOL. 8, NO. 6, December 1996.

[19] Daniel S. Tkach, "Information Mining with the IBM Intelligent Miner Familiy," *An IBM Software Solutions White Paper*, International Business Machines Corporation , February 1998

[20] Wen-Chi Hou, "Extraction and Applications of Statistical Relationships in Relational Databases," *IEEE Transaction on Knowledge and Data Engineering*, VOL. 8, NO. 6, December 1996.

[21] Daniel A. Keim, Hans-Peter Kriegel, "Visualization Techniques for Mining Databases: A Comparison," *IEEE Transaction on Knowledge and Data Engineering*, VOL. 8, NO. 6, December 1996.

[22] Surajit Chaudhuri and Umeshwar Dayal, "An Overview of Data Warehousing and OLAP Technology," VLDB Conference, 1996.

[23] Dan Bulos," OLAP Database Design A New Dimension," *Database Programming & Design*, June 1996.

[24] Rakesh Agrawal, Ashish Gupta, Sunita Sarawagi, "Modelling Multidimensional Databases," IBM Almaden Research Center, 1996

[25] Inmon, W.H., "Building the Data Warehouse", John Wiley, 1992.

[26] Josh Bersin, "Data Marts and Beyond: A Pragmatic Approach to Enterprise Decision Support," *Sybase, Inc*. 1997

[27] Information Discovery, Inc. "The Data Warehouse vs The Pattern Warehouse ™," *www.datamining.com*, 1998

[28] Wei-Min Shen, Bing Leng, "A Metapattern-Based Automated Discovery Loop For Integrated Data Mining -- Unsupervised Learning of Relational Patterns," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, no 6, pp. 898-910, Dec 1996.

[29] E.T. Jaynes, *Probability Theory-The logic of Science*. Cambridge University Press, 1997

[30] Wei-Min Shen, Bing Leng, "Metapattern Generation for Integrated Data Mining" The 2nd International Conference on KDD, Portland, Oregan, 1996.

[31] Gilles Brassard, Paul Bratley, Algorithmics Theory and practice. London, Prantice-Hall, Inc (1988)

[32] D. Maier, The Theory of Relational Databases. Rockville: Computer Science Press, 1983.

[33] A. Silberschatz and A. Tuzhilin, "What Makes Patterns Interesting in Knowledge Discovery Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, no 6, pp. 970-974, Dec 1996.

[34] John F. Rddick, Noel G. Craske, and Thomas J.Richards, "Handling Discovered Structure in Database Systems," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, no 2, pp. 227-240, April 1996.

[35] C. J. Date, *An Introduction to Database Systems*. Reading, Massachusetts Addison-Wesley (1995)

[36] Adnan Yazici and Mustafa İlker Sözat, "The Integrity Constraints for Similarity-Based Fuzzy Relational databases," *Int'l J. Intelligent Systems*, Vol. 13, pp. 641-659, 1998.

[37] M. Kantola, H. Mannila, K.-J. Raiha, and H.Siirtola, "Discovering Functional and Inclusion Dependencies in Relational Databases," *Int'l J. Intelligent Systems*, Vol. 7, pp. 591-607, 1992.

[38] H. Mannila and K.-J. Raiha, "Design by Example: An Application of Armstrong Relations," *J. Computer Systems Science*, vol. 40, no 2, pp. 126-141, 1986.

[39] Max Bramer, "Rule induction in data mining: concepts and pitfalls (2)," *Data Warehouse Report*, pp.22-27, autumn 1997.

[40] W. Kent, "The Universal Relation Revisited." *ACM TODS* 8, no 4, Dec 1983.