From: AAAI Technical Report WS-94-03. Compilation copyright © 1994, AAAI (www.aaai.org). All rights reserved.

Using Metaqueries to Integrate Inductive Learning and Deductive Database Technology

Wei-Min Shen, Bharat Mitbander, and KayLiang Ong
MCC, 3500 West Balcones Center Drive
Austin, TX 78759
{wshen, mitbander, ong}@mcc.com

Carlo Zaniolo
Computer Science Department, UCLA
Los Angeles, CA 90024
zaniolo@cs.ucla.edu

Abstract

This paper presents an approach that uses metaqueries to integrate inductive learning with deductive database technology in the context of knowledge discovery from databases. Metaqueries are second-order predicates or templates, and are used for (1) Guiding deductive data collection, (2) Focusing attention for inductive learning, and (3) Assisting human analysts in the discovery loop. We describe in detail a system that uses this idea to unify a Bayesian Data Cluster with the Logical Data Language (LDL++), and show the results of three case studies, namely: discovering regularities from a knowledge base, discovering patterns and errors from a large telecommunication database, and discovering patterns and errors from a large chemical database.

1 Introduction

Recent progress in knowledge discovery from databases (e.g., [2, 7]) has shown that inductive hypotheses generation (bottom-up) and deductive hypotheses verification (top-down) are both crucial components for effective discovery systems. Inductive learning is essential for generating hypotheses from data automatically, while on the other hand, deductive database technology is a natural tool for gathering evidence in support of existing hypotheses. However, most discovery systems, except perhaps Recon [5], do little towards integrating these components. Rather, the user often has to make a choice between a top-down or bottom-up approach at the outset, and the system makes no attempt to exploit the synergy between these two intrinsically related components.

In this paper, we propose an approach that addresses the integration of the inductive and deductive components via second-order predicates called *metaqueries*. A metaquery in essence is a predicate template. For example, in the metaquery $P(x\ y) \land Q(y\ z) \Rightarrow R(x\ z)$, P, Q, and R are variables that can be bound to any predicates. As shown in Figure 1, metaqueries serve as the link between the inductive and deductive aspects of knowledge discovery, effectively facilitating a deductive-inductive discovery loop. Metaqueries outline the data collecting strategy for the deductive part of the loop; they serve as the basis for the generation of specific queries which are obtained by instantiating the variables in the metaqueries with values representing tables and columns in the database of interest. These instantiated queries are then run against the

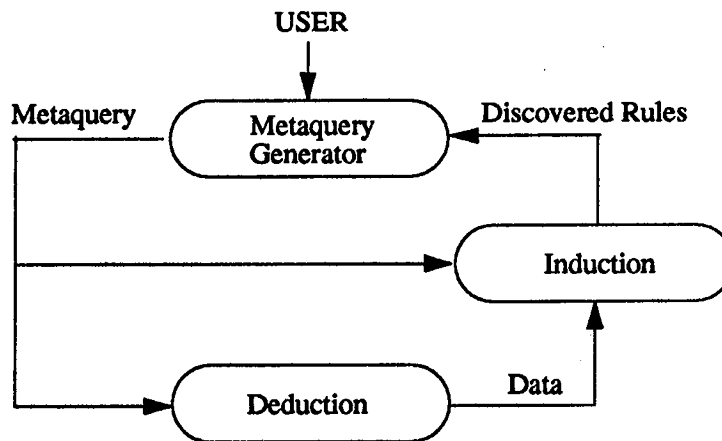

Figure 1: Using Metaqueries To Integrate Induction And Deduction In A Discovery Loop

database to collect relevant data. Similarly, metaqueries also serve as a generic description of the class of patterns to be discovered and help guide the process of data clustering and pattern generation in the inductive part of the loop. The patterns discovered from the database adhere to the format of the current metaquery. For example, one possible result of the above metaquery is citizen(x y)$\wedge$officialLanguage(y z) $\Rightarrow$speaks(x z), where "citizen," "officialLanguage," and "speaks" are relations that bind to P, Q, and R respectively in the current database.

Metaqueries can be specified by human experts or alternatively, they can be automatically generated from the database schema. Either way, they serve as a very important interface between human "discoverers" and the discovery system. Using metaqueries, human experts can focus the discovery process onto more profitable areas of the database; the system generated metaqueries provide valuable clues to the human expert regarding good start points for the database searches and also serve as the evolutionary basis for the development of user specified metaqueries more attuned to the discovery goals as envisaged by the human expert.

The above ideas are implemented in a system called the *Knowledge Miner* (KM). The induction part of KM is an unsupervised Bayesian Data Cluster, and the deductive part of KM is a state of the art deductive database technology called LDL++, which is well suited for both knowledge representation and data querying. The patterns or relations to be discovered are probabilistic implication rules. Users can either type their metaqueries directly, or have the system run automatically (i.e., let KM generate its own metaqueries).

In the rest of this paper, Section 2 gives an overview of the KM framework and its various components. Sections 3 and 4 describe in detail the LDL++ language and the unsupervised Bayesian Data Cluster, respectively. Sections 5 to 7 present three case studies in the application of this approach to knowledge discovery in a knowledge base, a telecommunication database, and a chemical database respectively. Finally, the paper is concluded with a set of open problems associated with this approach and our plan for future work.

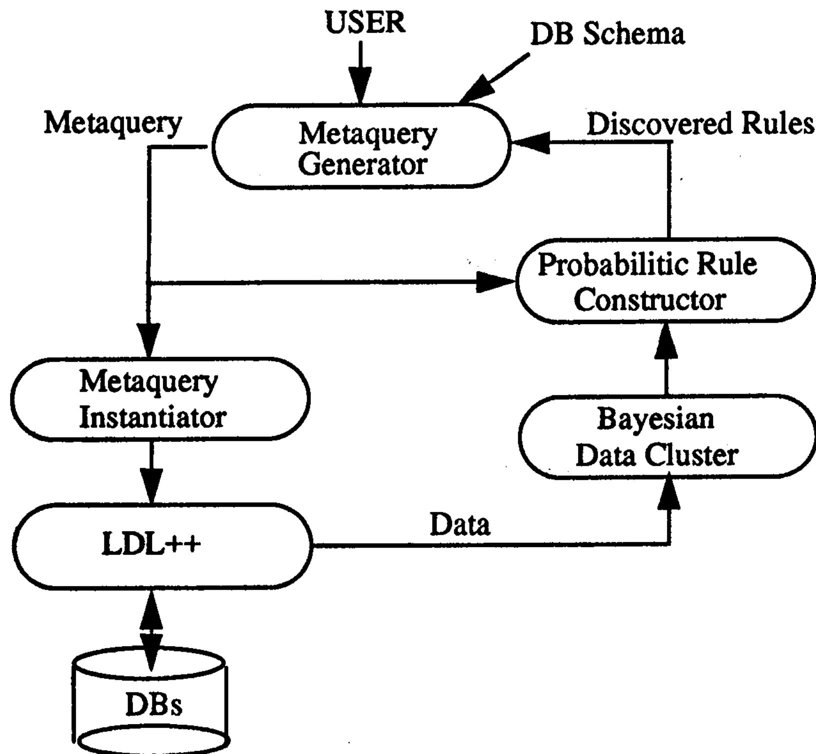

Figure 2: Knowledge Miner: A MetaQuery-Driven Framework

2 Knowledge Miner: A MetaQuery-Driven Framework

In this section, we give an overview of the metaquery driven framework, and a specific system called "Knowledge Miner" that implements the ideas described above.

Figure 2 illustrates the configuration of Knowledge Miner. Given a database and its schema, the metaquery generator suggests a series of valid metaqueries. Users can either pick some of the suggested metaqueries or they can specify their own metaqueries to be run.

On being provided with a metaquery as input, the instantiator generates a set of corresponding LDL++ queries by instantiating the predicate variables in the metaquery with relevant table names and column names as appropriate. These LDL queries are then run against the database to gather the data in the desired fashion. The resulting data is fed to the Bayesian Data Cluster, which classifies the data into classes and then passes the classification information to the Probabilistic Rule Constructor. The constructor then builds relations and patterns in the format specified in the current metaquery. The classification information produced by the Bayesian Data Cluster can also be stored persistently in a database.

Consider, for example, a database that has 4 tables, each with an arity of 2. Suppose that table1 and table2 specify ingredients of chemical compounds, and table3 and table4 specify properties of compounds (These facts can be presented to KM by experts or be extracted automatically from the schema by the metaquery generator). If we are interested in finding the relation between ingredients and properties, then a proper metaquery is as follows:

$$MQ[ingredients(X\,\mathrm{C1}\,\mathrm{C2}) \Rightarrow properties(X\,Y),\ valueCluster(Y)]$$

where X and Y are variables, C1 and C2 are constants, and *valueCluster* specifies a list of variables

whose values need to be clustered in the final results. Given this metaquery, the instantiator instantiates it into a series of LDL queries with varibles bound to appropriate table and column names. In the current example, such LDL queries are:

$$\text{table1}(X, \text{C1}), \text{table2}(X, \text{C2}) \Rightarrow \text{table3}(X, Y)$$
$$\text{table1}(X, \text{C1}), \text{table2}(X, \text{C2}) \Rightarrow \text{table4}(X, Y)$$

where C1 and C2 are constants, X, and Y are variables. Since *valueCluster* contains the variable Y, the system has to determine the probabilistic distribution of the values of Y for each LDL rule.

Each rule is run against the database, and a set of values that satisfy the constraints on the left side of the rule is fed into the Bayesian Data Cluster. The cluster classifies these values into a set of classes, each with a mean value and a variance. For example, when C1='BX89' and C2='GF102', the Y values that satisfy the first rule may be classified into two classes: (m_1, v_1, c_1) and (m_2, v_2, c_2), where m_i are mean values, v_i are variances, and c_i are the counts of values that fall into the class.

These results are then fed into the Rule Constructor, which uses the current metaquery as a template to generate the final discovered rules. For example, suppose the two classes generated from classification are: (m_1=2.4, v_1=3.5, c_1=388) and (m_2=202.0, v_2=0.5, c_2=12), then two rules constructed using the metaquery may look like these:

(P_1=0.97) ingredients='BX89'&'GF102' ⇒ Distribution(table3,column2)=(2.4, 3.5)
(P_2=0.03) ingredients='BX89'&'GF102' ⇒ Distribution(table3,column2)=(202.0,0.5)

where P_i is the probability of each rule, defined as $P_i = c_i / \sum_i c_i$.

We now discuss each of the components of KM in detail. In particular, the following two sections cover the LDL++ language and the Bayesian Data Cluster. The Metaquery Generator will be discussed after the presentation of experimental results obtained using KM.

3 The LDL++ Deductive Database System

The LDL++ system is a deductive database system based on the integration of a logic programming system with relational database technology. It provides a logic-based language that is suitable for both database queries and knowledge representation. More details on the LDL++ system and language can be found in [1, 6, 14, 15]. In this section, we will briefly describe some of the salient aspects of the LDL++ system as relevant to KM and also highlight the benefits of using LDL++ in the Knowledge Miner framework.

The LDL++ query language is based on Horn clause logic and a LDL++ program is essentially a set of declarative rules. For example, the following rules

$$\begin{aligned} ancestor(X,Y) &\leftarrow parent(X,Y). \\ ancestor(X,Y) &\leftarrow ancestor(X,Z), parent(Z,Y). \end{aligned}$$

specify that a new relation ancestor/2 can be defined based on the relation parent/2. X, Y and Z are variables and ancestor and parent are predicate symbols. By declarativeness, we mean that the ordering between the rules is unimportant and will not affect the results returned. Deduction of all values of ancestor/2 is achieved through an iterative bottom-up execution model.

The language supports a rich set of complex data types such as complex objects, lists and sets in addition to other basic types such as integer, real and string. Examples of these complex types are *rectangle(1,2)*, *[1,2]* and *1,2* respectively.

With such a rich set of data types, the LDL++ language is well suited for representing knowledge discovered from databases. Furthermore, complex constraints in the metaqueries can be easily

specified in the language. Through the metalevel query facility in LDL++, constraints represented in the form of data can be used to generate rules at run-time.

From the database query perspective, in comparison to SQL, the LDL++ language extends the ability of KM to impose more complex constraints on the data it retrieves beyond what is possible in SQL. This is illustrated by the recursive rules that define ancestor/2 - such constraints cannot be specified in SQL directly. Moreover, cascading or nested queries can be easily represented in LDL++ in the form of rules.

Systemwise, the open architecture of the LDL++ system meets many of the KM demands. It is "open" to procedural languages such as C/C++ in two ways: It provides an Application Programming Interface (API) that allows applications to drive the system and, an External Function Interface (EFI) that allows C/C++ routines to be imported into the inference engine. It is also "open" to external databases such as Sybase, Oracle, Ingres and DB2 through its External Database Interface (EDI).

Referring to Figure 2, we can observe that the LDL++ system has to interface with the Instantiator, the Database and the Bayesian Data Cluster. It interfaces with the Instantiator throught its API, the Bayesian Data Cluster with its EFI and the Database with its EDI.

Both, tables in the external databases and C/C++ interface routines, are modeled as predicates through its EDI and EFI respectively. As a result, these external resources are transparent to the inference engine and the KM can plug in a different database or another learning algorithm with ease, without making any changes to the overall implementation. The EDI facilitates the retrieval of data as well as the archiving of statistical metadata into persistant storage.

The EDI and EFI are also convenient for gathering data from multiple, heterogenous databases or files. This empowers KM to discover relationships between data from different sources. Again, the data access is transparent to KM and no changes are required to take advantage of this capability.

Last but not least, once the statistical data is derived and high-probability rules are created, they are represented directly in LDL++ for enforcing the integrity of the database.

4 Bayesian Data Clustering

To classify the retrieved data, we use a *conceptual clustering* approach based on the Cobweb algorithm [3] but adapted to use discriminant functions derived from Bayesian probability theory [4, 8]. We feel that the latter lends a stronger mathematical basis to the evaluation process.

The Cobweb algorithm facilitates incremental, unsupervised learning of concept hierarchies from attribute-based instances. The concepts formed represent nodes in a concept tree, with each non-leaf node further partitioned into its more specific children nodes. Thus, more general concepts are located near the root of the tree and more specific concepts are located towards the leaves. Each node in the concept tree describes a class of instances and hence each cut of the concept tree represents a partition of all the known instances.

Given a new instance, the task of the Cobweb algorithm is to create a new concept hierarchy that incorporates the knowledge embodied in the instance. The classification of the new instance proceeds recursively through each level of the concept hierarchy, following a path composed of increasingly specific concept nodes. At each level in the concept hierarchy, the classification process is focussed on the immediate subtree (i.e. the children) of the current node and essentially reduces to the best choice among the following four alternatives, namely:

- *Incorporate* the instance into an existing concept. Select the child concept which best fits the new instance to host it.

- *Create* a new child concept to host the new instance.
- *Merge* two children concepts and use the resulting combined concept to host the new instance.
- *Split* a child concept i.e. replace it with its children and then choose the best child to host the instance.

The best choice is determined in our case via the application of the *Bayesian Evaluation Function*, which is described a little later in this section. The algorithm recursively applies the above classification process at each level of the concept hierarchy, incorporating the instance into concepts of increasing specificity. The algorithm halts when it classifies the instance into either:

- A singleton node at the leaf, or
- A new concept at any level in the hierarchy.

In the Knowledge Miner framework, the Bayesian Data Cluster interfaces with the LDL++ module through the latter's EFI. It receives the data extracted by LDL++ as input, classifies it and returns the concept tree as output. The module interface provides all the requisite functionality to the external process/user for fine-tuning the classification process.

4.1 The Bayesian Evaluation Function

An instance is described by its attributes. Thus, if K is the number of attributes, an instance D can be defined as a vector of K values for these attributes:

$$D \equiv (x_1 = v_1, ..., x_k = v_k, ..., x_K = v_K)$$

A concept is a collection of related or similar instances. The size of a concept C, denoted as $|C|$, is the number of its instances. The concept C can be defined as a vector of K probability distribution functions on the K attributes of its instances:

$$C \equiv (f_1, ..., f_k, ..., f_K).$$

where f_k is determined by the kth values of all its instances. For instance, if $v_{k1}, v_{k2}, \ldots, v_{kN}$ are the kth values of N instances $D_1, D_2, \ldots, D_N$, respectively, then, assuming a normal distribution, the probability distribution function f_k can be estimated as:

$$f_k(x_k|D_1 \cdots D_N I) = M_k \exp\left\{-\frac{(x_k - a_k)^2}{2\sigma_k^2}\right\} \tag{1}$$

where I is the background information

$$M_k = \frac{1}{\sigma_k\sqrt{2\pi}},$$

is a normalization constant to satisfy the constraint:

$$\int_{-\infty}^{+\infty} f_k(x_k|D_1 \cdots D_N I)dx_k = 1,$$

a_k, the mean of f_k, can be estimated as:

$$a_k = \frac{1}{N}\sum_{j=1}^{N} v_{kj} \tag{2}$$

and σ_k^2, the variance of f_k, can be estimated as:

$$\sigma_k^2 = \frac{1}{N}\sum_{j=1}^{N}(v_{kj} - a_k)^2. \tag{3}$$

The values of a_k and σ_k^2 can also be estimated incrementally for the nth instance based on the values of the same for the $n-1$th instance - see [12] for more details. The incremental values are estimated as:

$$a_{kN} = a_{k(N-1)} + \frac{v_{kN} - a_{k(N-1)}}{N} \tag{4}$$

and

$$\sigma_{kN}^2 = \frac{N-1}{N}\left[\sigma_{k(N-1)}^2 + (a_{k(N-1)} - a_{kN})^2\right] + \frac{(v_{kN} - a_{kN})^2}{N} \tag{5}$$

Let $H = (C_1, \ldots, C_J)$ represent the children of concept C. Let child C_j $(1 \leq j \leq J)$ be the one selected to host new instance D, and let H_j represent the new set of children nodes obtained after the definition of C_j changes to incorporate D. Then how good the assignment of D to C_j is can be estimated by the probability:

$$P(H_j|DH) = P(H_j|H)\frac{P(D|H_jH)}{P(D|H)} \tag{6}$$

Clearly, the best child concept to host D is C_i such that

$$P(H_i|DH) = \max_{j=1,\ldots,J}\{P(H_j|DH)\} \tag{7}$$

Assume that $P(H_j|H)$ are equal for all j, then to compare $P(H_j|DH)$ we need only compute $P(D|H_jH)$. Notice that

$$P(D|H_jH) = \sum_{j=1}^{J} P(DC_j|H_jH) = \sum_{j=1}^{J} P(C_j|H_jH)P(D|C_jH_jH) \tag{8}$$

and

$$P(C_j|H_jH) = \frac{|C_j|}{|C|} \tag{9}$$

and

$$P(D|C_jH_jH) = P(D|C_j) \tag{10}$$

$P(D|C)$ represents the degree to which instance D belongs to concept C and is computed as:

$$P(D|C) = \prod_{k=1}^{K} P(v_k|f_k) \tag{11}$$

The term $P(v_k|f_k)$ can be computed easily. If the kth attribute is numeric, then

$$P(v_k|f_k) \approx f_k(v_k)\Delta x_x \tag{12}$$

where Δx_k is a small constant range around v_k. If the kth attribute is discrete, then

$$P(v_k|f_k) = \frac{\text{num. of insts whose } k\text{th value is } v_k}{\text{num. of insts in } C} \tag{13}$$

5 Discoveries from A Large Knowledge Base

In this and the following two sections, we present the results of three applications of Knowledge Miner to a large common-sense knowledge base called Cyc, a telecommunication database containing a large number of real telephone circuits and a chemical research database representing over 30 years of chemical research results respectively.

Cyc is a large common-sense knowledge base developed at MCC, containing over one million logical assertions (think of them as relations if you like). We applied an earlier version of the Knowledge Miner system [10, 11] to two large collections of objects in Cyc: *Person* and *Organization*. At the time, the *Person* collection contained 633 objects and 77 relevant relations, while the *Organization* collection contained 1550 objects and 150 relevant relations.

The metaquery we used in these experiments is as follows:

$$MQ[P(x\ y) \wedge Q(y\ z) \Rightarrow R(x\ z),\ valueCluster()]$$

where P, Q, and R are variables for relations, and x, y, and z are variables for objects. Note that *valueCluster* is empty, as we are only interested in the relation $P(x\ y) \wedge Q(y\ z) \Rightarrow R(x\ z)$ and not the value clusters of x or z.

This metaquery is instantiated on combinations of the 77 relations in the *Person* collection and the 150 relations in the *Organization* collection respectively. From the data in Cyc, 146 regularities were discovered (i.e. were found to have enough supporting data) from the *Person* collection, and 250 from the *Organization* collection. Example regularities discovered from the *Person* collection are as follows:

```
acquaintedWith(x y)∧languageSpoken(y z)⇒languageSpoken(x z)
studentAtInstitution(x y)∧hasProfessor(y z)⇒likedBy(x z)
primeMinisterOfCountry(x y) ∧ eq(y z) ⇒ headOfGovernmentOf(x z)
computersFamiliarWith(x y)∧languagesThisFamilyOfMachinesCanRun(y z)⇒programsIn(x z)
presidentOfCountry(x y)∧officialLanguage(y z)⇒languageSpoken(x z)
likedBy(x y)∧wearsClothing(y z)⇒wearsClothing(x z)
birthDate(y z)∧allSuperAbstrac(x y)⇒birthDate(x z)
laterSubAbstractions(x y) ∧ eq(y z) ⇒ startsBeforeStartingOf(x z)
children(x y) ∧ eq(y z) ⇒ relative(x z)
duringActorIn(x y) ∧ eq(y z) ⇒ canPerform(x z)
isAwareOf(x y) ∧ allSubAbstrac(y z) ⇒ isAwareOf(x z)
residences(x y)∧officialLanguage(y EnglishLanguage)⇒speaks(x EnglishLanguage)
memberOfPoliticalParty(x y)∧allInstances(y CommunistParty)⇒ideology(x Communist)
staffMembers(x y)∧eq(y MCC)⇒probableExpertiseInComputers(x UserLevel)
```

All of these assertions were new to Cyc. Most of these assertions are interesting because they specify things that are true with high, but less than one, probabilities. For example, the regularity studentAtInstitution(x y)∧hasProfessor(y z)⇒likedBy(x z) says that if x is a student in an institution, then x must be liked by all the professors in that institution. This is not always true but is a good default rule. Given the fact that more and more knowledge bases are capable of reasoning nonmonotonically, this kind of default knowledge is important if you don't know whether a particular student x is liked by a particular professor z.

6 Discoveries from A Telecommunication Database

We also applied KM to a large telecommunication database. This database contains approximately 300MB of information about existing telephone circuits in a regional Bell company. There are two types of circuits in the database: point to point circuits, and multiple-leg circuits. Each circuit (or leg) is a sequence of connected components such as *channel, cable, station, equipment, test point,*

etc.. Each component is associated with a set of features. For example, an *equipment* has location, function, channelID, and others. A *channel* has starting and ending locations, conductivity, and others.

We were interested in finding the feature correlation between components that are directly connected in some circuit. So the metaquery used was as follows:

$$MQ[linked(X\ Y) \Rightarrow equal(feature1(X), feature2(Y)),\ valueCluster()]$$

where X and Y are variables for components. From this metaquery, the instantiator generated a series of LDL queries, with *linked* bound to a circuit connection table, and *feature1* and *feature2* bound to various columns in each component table. Running these LDL queries against the database, the KM system generated 281 patterns in about 7 hours. Two examples of patterns that have high probabilities are:

(P=1.0) linked(EQPT,EQPT)⇒equal(chanID(EQPT),chanID(EQPT))
(P=0.98) linked(CHANNEL,EQPT)⇒equal(endLocation(CHANNEL),location(EQPT))

Two examples of patterns that have low probabilities are:

(P=0.015) linked(LOOP,CABLE)⇒equal(startLocation(LOOP),endLocation(CABLE))
(P=0.004) linked(EQPT,CHANNEL)⇒equal(calcSigVoice(EQPT),ovrdSigVoice(CHANNEL))

Two examples of patterns that have middle probabilities are:

(P=0.59) linked(TPoint,TPoint)⇒equal(accsys(TPoint),accsys(TPoint))
(P=0.40) linked(EQPT,TPoint)⇒equal(hierchyIND(EQPT),analRingSigDir(TPoint))

We presented these rules to experts in this field and most of the rules were demmed to make perfect sense. There are a number of ways that these rules can be used: Those with high probabilities can run directly (they are in fact LDL queries) against the database to find the set of data that violate the rules. Such data is very likely to be erroneous. Similarly, one can run the low probability rules to find the data that obeys the rules - this data is also very likely to be erroneous. These rules can also be used as data validation filters for future data entry. Any input data that violates the high-probability rules (or obeys the low-probability rules) should be flagged and checked before actually being entered into the database. The rules with middle probabilities are also interesting and useful in that they identify subsets of the database that need further investigation. One can then introduce new metaqueries that are constrained to focus on these subsets and subsequently generate new rules with high and low probabilities.

7 Discoveries from A Chemical Research Database

We are currently applying the KM system to a real chemical research database. This database contains results representing over 30 years of chemical research and experiments. Conceptually speaking, it has information about the ingredients and the properties of various compounds. The ingredients are specified in a single table, while the properties are distributed over more than 50 tables. All these tables are linked via the compound identification.

We are interested in discovering the relationship between ingredients and properties, so the following metaquery is employed:

$$MQ[ingredient(C\ X\ Y) \Rightarrow properties(C\ Z),\ valueCluster(Z)]$$

where C is a variable for compound identification, X and Y are variables for ingredients, and Z is a varible for property. As one may expect given the large number of properties in the database,

this metaquery is too time consuming to run. So we restrict it by constraining the variables X and Y to be bound only to those ingredients that are commonly used.

Using the restricted metaquery, KM returns a set of patterns describing the relationships between specific ingredients and properties. One example of these patterns is the relationship between an ingredient *3060* and the *density* property of the compound:

(P=0.99) ingredient(C,3060,Y) $\Rightarrow$Distribution(density,BM)=(1.243,0.46)
(P=0.01) ingredient(C,3060,Y) $\Rightarrow$Distribution(density,BM)=(-999.9,0.2)

The first rule indicates that the majority of values of (density,BM) are near 1.243 (with a variance 0.46). Thus, the ingredient *3060* has a very specific effect on the compound's *density BM* property, irrespective of the other ingredient Y. This information is extremely valuable to the chemists. The second rule indicates that there is some noisy data in the density table.

We are currently experimenting with various metaqueries, and the results will be reported in future papers.

8 Generating Initial Metaqueries

As we have seen so far, metaqueries play an important role in the KM discovery system. However, good metaqueries do not come easy. They must balance the computational expense on one hand with the potential profit of discovery on the other. One may have an extremely general metaquery for a database, but it's demands on computational resources may be unteneble. On the other hand, one may have an extremely specific metaquery with fast response characteristics, but it may not discover anything interesting. The ideal case would be to have a metaquery general enough to support the discovery of interesting patterns, but constrained enough to do so without making unreasonable demands on computational and time resources.

Obviously, designing the right metaquery is a difficult problem - in addition to knowledge about the database schema, one also needs good judgement and a healthy dose of intuition. We do not claim to have reduced this process to an exact science - indeed, we have not solved this problem. Instead, our philosophy, as reflected in the KM methodology, is to play the role of an informed assistant and afford a human expert the fullest flexibility and help in defining the best metaqueries he or she can. For example, one thing we find useful is to have the system automatically generate a set of initial metaqueries, which can then be analyzed and refined by the human expert. As more rules are discovered from the database, the system should be able to generate more metaqueries in an interactive fashion. This is the task of the Metaquery Generator in KM.

The Metaquery Generator accepts two inputs: the database schema and the rules discovered from the database. Initially, the generator examines the database schema and generates simple metaqueries that address the relationships across tables. Certain types of information from the schema can provide useful guidance in narrowing down the set of possible metaqueries. These include:

1. Keys
2. Column Types
3. Column Names

Some example heuristics based on these informaton types include:

- Since the key column of a table is always unique, we can pretty much eliminate it from consideration as a possible candidate for instantiation in metaquery.

- When joining any two tables, we need to determine the join columns i.e. a column from each table which could be deemed equal. In this case, the basic criterion is that both columns must have the same types to even have a reasonable chance at success.

- If the column names are the same, the likelihood is greater that the two tables should be joined based on those two columns.

Discovered rules can also be used to guide the generation of metaqueries. For example, knowing the probabilistic distribution of the candidate columns for joining two tables, we can determine whether a metaquery based on the same will generate useful results or not. Thus, less useful metaqueries will not be suggested to the users. Another way to generate metaqueries is to generalize the existing rules. For example, the system BLIP [13] uses the heuristic of "turning constants to variables" to acquire metalevel rules from domain-level rules entered by users.

The Metaquery Generator is by no means complete. We have yet to generate complex queries automatically. There is also potentially a lot more useful information that can be gleaned from the database schema and the discovered rules that could be gainfully applied to the generation process. This is ongoing work and we are engaged in a little discovery activity ourselves - we will have more to report as more experiements are carried out.

9 Conclusions and Future Work

We have presented a framework for using metaqueries to integrate induction and deduction for knowledge discovery from databases. The framework has been implemented in the Knowledge Miner system to integrate two very advanced deductive and inductive technologies: the LDL++ system and the Bayesian Data Cluster. Applications of Knowledge Miner to several real-world databases and a knowledge base have demonstrated that the metaquery approach is indeed viable and shows much promise.

Our experiments with the system thus far have been fruitful, but a lot more work needs to be done before the KM system can be truly useful in a variety of discovery environments. On the theoretical side, we are exploring more advanced learning algorithms with a better, more intelligent clustering capability as well as a higher efficiency in terms of computational time and space used.

On the practical side too, there are a number of issues on the agenda. One is to have the ability to connect with legacy databases provided by various vendors. The LDL++ system can currently access data from Sybase, Ingres, Rdb and Oracle. We plan to extend it to also access other database products such as IBM/DB2 and IBM/IMS. A second issue is the design and implementation of an appropriate visualization tool for the graphical presentation of the results of the discovery process.

Lastly, we will also explore new uses for the discovered rules and statistical data. In addition to using them to enforce integrity constraints for future updates and for generating new metaqueries, we believe that they can also be effectively used for estimating the cost of queries and also for supporting some forms of fuzzy querying. These beliefs remain to be verified.

References

[1] Arni, Ong, Tsur, and Zaniolo. 1994. LDL++: A Second Generation Deductive Database System, Working Paper.

[2] Cercone, N. and M. Tsuchiya. Guest editors. 1993. Special issue on Learning and Discovery in Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6).

[3] Fisher, D. H., 1987. Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, 2:139-172.

[4] Jaynes, E. T. 1992. *Probability Theory — The Logic of Science*, Washington University. St. Louis, MO.

[5] Kerber, R. and B. Livezey and E. Simoudis. 1994. Recon: A Framework for Database Mining. *International Journal of Intelligent Information Systems.* Forthcoming.

[6] Naqvi and Tsur. 1989. A Logical Language for Data and Knowledge Bases. W. H. Freeman Company.

[7] Piatetsky-Shapiro, G. Editor. 1993. Proceedings of AAAI-93 Workshop on Knowledge Discovery in Databases. AAAI Press.

[8] Shen, W. M. 1993. *Bayesian probability theory — A general method for Machine Learning*, MCC Technical Report, Carnot-93-101.

[9] Shen, W. M. 1994. *Autonomous Learning from the Environment.* Computer Science Press. W.H. Freeman.

[10] Shen, W. M. 1991. Discovering Regularities from the Cyc Knowledge Base. *Proceedings of the 8th International Conference on Machine Learning.*

[11] Shen, W. M. 1992. Discovering Regularities from Knowledge Bases. *International Journal of Intelligent Systems*, 7(7), 623-636. Special issue on Knowledge Discovery in Databases and KnowledgeBases (Selection of best papers from KDD-91 workshop). G. Piatetsky-Shapiro, guest editor.

[12] Shen, W. and B. Mitbander and K. Ong. 1993. A Bayesian Cobweb Algorithm for Unsupervised Data Clustering. MCC Technical Report Carnot-025-94.

[13] Theime, S. 1989. The Acquisition of Model-Knowledge for A Model-Driven Machine Learning Approach. In Morik, K. ed. *Knowledge Representation and Organization in Machine Learning.* Springer-Verlag.

[14] Tsur, Arni, and Ong. 1993. The LDL++ User Guide, MCC Technical Report Carnot-012-93(P).

[15] Zaniolo, C. 1992. Intelligent Databases: Old Challenges and New Opportunities. *Journal of Intelligent Information Systems*, 1. 271-292. Kluwer Academic.