Intelligent Query Answering by Knowledge Discovery Techniques *

Jiawei Han[†] **Yue Huang**[†] **Nick Cercone**[†‡]

[†] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6, e-mail: {han, yhuang, nick}@cs.sfu.ca

[‡] Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada S4S 0A2, e-mail: nick@cs.regina.ca

Abstract

Knowledge discovery in databases facilitates querying database knowledge, cooperative query answering and semantic query optimization in database systems. In this paper, we investigate the application of discovered knowledge, concept hierarchies, and knowledge discovery tools for intelligent query answering in database systems. A knowledge-rich data model is constructed to incorporate discovered knowledge and knowledge discovery tools. Queries are classified into data queries and knowledge queries. Both types of queries can be answered directly by simple retrieval or intelligently by analyzing the intent of query and providing generalized, neighborhood or associated information using stored or discovered knowledge. Techniques have been developed for intelligent query answering using discovered knowledge and/or knowledge discovery tools, which includes deduction, generalization, data summarization, concept clustering, rule discovery, query rewriting, lazy evaluation, semantic query optimization, etc. Our study shows that knowledge discovery substantially broadens the spectrum of intelligent query answering and may have deep implications on query processing in data- and knowledge-base systems.

Index Terms – Database and knowledge-base systems, knowledge discovery in databases, knowledge-rich data model, intelligent query answering, semantic query optimization, query analysis and query processing.

1 Introduction

Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [5]. The growth in the size and number of existing databases far exceeds human abilities to analyze such data, thus creating both a need and an opportunity for extracting knowledge from databases. Knowledge discovery in databases has become an active area in both database and machine learning research [22, 19].

With the rapid development of knowledge discovery techniques, it is natural to study the applications of the technology in querying database knowledge and processing queries in database systems. In this paper, we investigate the application of discovered rules, concept hierarchies and knowledge discovery techniques to intelligent query answering in database systems. The study is based on the premise that efficient, relatively sophisticated knowledge discovery tools will soon become available for large database systems. Such a view has been reinforced by some preliminary studies and experimental results [5, 19, 6]. For example, in our previous studies [1, 6], an attribute-oriented induction method has been developed for knowledge discovery in databases. The method integrates a machine learning paradigm, especially *learning-from-examples* techniques, with database operations and provides an efficient way for extraction of generalized data from actual data in databases. Other studies, such as [5, 11, 19], have also demonstrated the promising future of knowledge discovery in databases.

To study query processing in a knowledge-rich database associated with knowledge discovery tools, it is often necessary to distinguish data, knowledge and queries defined at the primitive data level from those defined at a high concept level. Data in a knowledge-rich database are classified into **primitive data** and **high-level data**. The former are actual data stored in data relations and, if appearing in some concept hierarchies, correspond to the primitive level (i.e., leaf) nodes of the hierarchies; whereas the latter are nonprimitive data subsuming primitive ones and residing at the nonprimitive level of concept hierarchies. Correspondingly, a **primitive-level query** is a query whose constants involve only primitive data; whereas a **high-level query** is a query whose constants involve

*The work was supported in part by the research grants from the Natural Sciences and Engineering Research Council of Canada and a research grant from the Centre for Systems Science of Simon Fraser University. The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada and the participation of PRECARN Associates Inc.

high-level data. Similarly, rules (notice that *integrity constraints* can be viewed as a special kind of rule) can be classified into **primitive-level** and **high-level rules** based on their reference to high-level data.

Many deduction rules in *traditional* deductive databases are primitive-level rules because they do not reference high-level data. However, a deduction rule can also be a high-level one if it references high-level data. For example, "*every young faculty member in computing science has a Ph.D. degree in the field*" is a high-level rule if the concepts "*young*" and "*faculty*" are high-level concepts.

Since a rule can be defined explicitly or be extracted by a knowledge discovery process, it is important to differentiate a deduction rule from a generalized one. *A rule which is explicitly defined by a user or an expert* is a **deduction rule**; whereas *a rule which is generalized from a database state* is a **generalized rule**. Both deduction rules and generalized rules can be primitive-level or high-level rules. However, since a generalized rule summarizes data from a database state, it reflects a *general fact* in the current database state but does not enforce a *constraint* on the possible database states. This contrasts with a deduction rule or an integrity constraint which states a rule (or a constraint) that a *potential* database state *must* follow. For example, a generalized rule, "*all of the teaching assistants are graduate students*", states a fact in the current database but does not claim that all the teaching assistants *must be* graduate students nor reject the *possibilities* of hiring an undergraduate student as a teaching assistant. However, the same rule, if stated as a deduction rule or an integrity constraint, will reject the possibility of allowing an undergraduate student to serve as a teaching assistant in a consistent database.

In a knowledge-rich database system, there may exist two kinds of queries, *data queries* and *knowledge queries*, where a **data query** is to *find concrete data stored in databases, which corresponds to a basic retrieval statement in a database system*; whereas a **knowledge query** is to *find rules and other kinds of knowledge in the database, which corresponds to querying database knowledge [16] including deduction rules, integrity constraints, generalized rules and other regularities.* For example, "*retrieving all of the students who took the course CMPT-459 in 1992*" is a data query; whereas "*describing the general characteristics of those students*" is a knowledge query. Furthermore, it is often desirable to provide intelligent and assisted answers to queries *besides* (or *instead of*) direct retrieval of data and knowledge. Thus, query answering mechanisms in a knowledge-rich database can be classified based on their responses to queries into two categories: *direct query answering* and *intelligent (or cooperative) query answering.* **Direct query answering** is *a direct, simple retrieval of data or knowledge from the knowledge-rich database*; whereas **intelligent query answering** consists of *analyzing the intent of query and providing generalized, neighborhood or associated information relevant to the query* [4]. For example, simple retrieval of the names of the students who take the designated course is direct query answering to the above data query; whereas summarizing the characteristics of those students, such as "*90% of them majored in computing science and took CMPT-359 as prerequisites*", provides an intelligent answer to the same data query.

In many cases, a database user may not be able to distinguish between primitive and high-level data and between information that is data and information that is knowledge. A knowledge query can often be viewed as a *follow-up* to a data query when the answers to a data query requires further explanation, reasoning or summarization. Therefore, it is important to provide a single, coherent framework to handle data and knowledge queries and to handle direct query answering and intelligent query answering. There have been some interesting studies on querying database knowledge and intelligent query answering [16, 21, 20, 10, 4]. Previous studies emphasize the application or inquiry of deduction rules and integrity constraints in relational or deductive databases. With the availability of generalized knowledge and knowledge discovery tools, queries can be posed and answered at levels higher than that of primitive concepts, and knowledge about general characteristics of data can be inquired or utilized in the processing of data or knowledge queries.

In this paper, we extend the domain of our study from a deductive database to a knowledge-rich database assisted with generalized knowledge and knowledge discovery tools. A knowledge-rich data model is constructed which consists not only of the components from a deductive database (including database schemas expressed by an extended deductive entity-relationship data model, data relations, deduction rules and integrity constraints) but also the components relevant to knowledge discovery processes, including concept hierarchies, generalized knowledge, and knowledge discovery tools. The knowledge discovery tools are used to extract general knowledge dynamically, when necessary, from any set of interested data in the database. A unified framework is established for answering data and knowledge queries in a knowledge-rich database. A systematic study is performed on intelligent query answering of both data or knowledge queries in a database system associated with discovered knowledge and knowledge discovery tools.

The paper is organized as follows. In Section 2, a data model is constructed for knowledge-rich databases, which consists of both deductive database and knowledge discovery components. In Section 3, we examine four basic categories of query answering in a knowledge-rich database according to the combinations of *data* vs. *knowledge* queries and *direct* vs. *intelligent* query answering mechanisms. In Section 4, we study knowledge discovery techniques and their role in intelligent query answering and query optimization. A discussion of the issues of intelligent query answering and a summary of our study are provided in Section 5.

2 A Data Model for Knowledge-Rich Databases

As an extension to the logic data model proposed in deductive database research [24], a *knowledge-rich data model* is constructed for databases with both deduction and knowledge discovery capabilities.

Definition 2.1 A **knowledge-rich database** (*KDB*) consists of six components: (1) *Schema*, a knowledge-rich database schema; (2) *EDB, an extensional database; (3) IDB, an intensional database; (4) H, a set of concept hierarchies; (5) GDB, a generalized database*; and *(6) KDT, a set of knowledge discovery tools*, defined as follows. $KDB = \langle Schema,\ EDB,\ IDB,\ H,\ GDB,\ KDT \rangle$.

1. **Schema**, a *knowledge-rich database schema*, describes the general structure and organization of KDB including (i) *physical* and *virtual* entities, attributes and relationships, and (ii) the organization of rules, integrity constraints and concept hierarchies, based on a deductive entity-relationship data model [7].
2. **EDB**, an *extensional database*, consists of a set of predicates, each corresponding to an extensional data relation.
3. **IDB**, an *intensional database*, consists of a set of *deduction rules* and *integrity constraints* (*ICs*).
4. **H**, a set of *concept hierarchies*, specifies taxonomies of concepts on top of primitive data in extensional and intensional databases.
5. **GDB**, a *generalized database*, consists of a set of *generalized rules* which summarize the regularities of the data at a high level.
6. **KDT**, a set of *knowledge discovery tools*, performs knowledge discovery efficiently in databases, when necessary.

□

The first component, *Schema*, follows from a deductive entity-relationship data model [7] which extends an entity-relationship model [3, 23] to incorporate rules, integrity constraints and complex data objects for deductive databases. The second and third components, *EDB* and *IDB*, are the same as in deductive databases [24] except that IDB rules can be defined by some nonprimitive data as well. Notice that a rule (or an integrity constraint) in the IDB can be first discovered by a knowledge discovery process and then be *recognized* and stored in the IDB as a regular rule or integrity constraint. However, once a *discovered* regularity is *recognized* and stored, it will play the same role as the *originally defined* one. Thus, we assume that all of the rules in IDB are defined ones.

The last three components, *H, GDB* and *KDT*, are the newly introduced knowledge discovery components which are used to incorporate discovered knowledge and knowledge discovery in databases.

H, a set of *concept hierarchies*, represents the relationships among concepts at different levels. The information about concept hierarchies can be *provided* by knowledge engineers or domain experts or be *discovered* automatically or semi-automatically using knowledge discovery tools based on the statistics of data distribution in databases and the relationships among different attributes [8]. Many concept hierarchies are implicitly stored in the database. For example, the hierarchical relationship among "*city*", "*province*" and "*country*" attributes are usually stored in the database and can be made explicit at the schema level by indicating a part-of-hierarchy, "$city \subset province \subset country$". It is realistic to have some concept hierarchies provided by knowledge engineers or domain experts even in a large database system since a concept hierarchy registers *only* the *distinct* discrete attribute values or *ranges* of numerical values for an attribute, which is, in general, not very large. Further, by providing different concept hierarchies, users or experts may have preference to control the knowledge discovery or intelligent query answering processes.

GDB, the *generalized database*, is another important component in the knowledge-rich database. Since there are usually a very large set of generalized rules which can be extracted from any interesting subset of data in a database by performing generalization in different directions, it is unrealistic to store all of the possible generalized rules. However, it is often useful to store some generalized rules or intermediate generalized relations in the GDB based upon the importance of the knowledge and the frequency of inquiries. The stored generalized rules are useful for querying database knowledge and semantic query optimization. Notice that a stored generalized rule should be incrementally updated after the updates of the relevant data set in order to preserve its correctness. This can be performed by an incremental learning algorithms provided in knowledge discovery tools [5, 6].

The last component, *KDT*, consists of a set of *knowledge discovery tools*, which could be a set of knowledge discovery algorithms or a database-oriented knowledge discovery subsystem, such as INLEN [11], KDW++ [5], DBLEARN [6], etc. Since a knowledge-rich database stores only a small portion of all of the possible generalized knowledge, it is often necessary to evoke a knowledge discovery process dynamically and extract general regularity from a specific set of data relevant to the query. The KDT tools can be used for on-line knowledge discovery and intelligent query answering.

The *university* database presented in Example 2.1 is an illustrative example of such a knowledge-rich database.

Example 2.1 Let a university database be modeled by a deductive entity-relationship model in which the extensional database (EDB) is mapped to a relational-like schema presented in Figure 1, where *Cnum* stands for *course number*, *TA* for *teaching assistant*, and *GPA* for *grade point average*.

Course (*Cnum, Title, Semester, Department, Instructor, TA, Enrollment, Time*).
Professor (*Pname, Department, Salary*).
Student (*Sname, Status, Sex, Major, Birth_date*(*Day, Month, Year*),
Birth_place(*City, Province, Country*), *GPA*).
Grading (*Student, Course, Grade*).

Figure 1: Schema of the University database

The concept hierarchies defined in the database are shown in Figure 2. The first three lines imply that the primitive data for *Status* is {*freshman, ..., Ph.D.*}, and their corresponding high-level data is *undergraduate* or *graduate* respectively. The entry "*Birth_place*(*City* ⊂ *Province* ⊂ *Country*)" indicates that the concept hierarchy for the attribute *Birth_place* is given by the data stored in the relation *Student* according to the part_of hierarchy: *City*, *Province* and *Country*. For example, a tuple,

$$Student(Tom\_Jackson, \ldots, Birth\_place(Vancouver, BC, Canada), \ldots),$$

indicates that *Vancouver* is a part of *British Columbia* (*BC*), which is in turn a part of *Canada*, in the concept hierarchy for *Birth_place*.

Notice that there are many different kinds of hierarchical relationships among data in a database, such as part_of, is_a, subset_of, etc., which may play different roles in conceptual analysis. The different semantics among concept hierarchies are not essential in the knowledge discovery algorithm itself since different concepts are generalized to their corresponding higher level concepts by following their corresponding concept hierarchies in a similar manner in the generalization process. However, such semantic differences will be important in the analysis of query intent and provision of intelligent answers.

{ freshman, sophomore, junior, senior } ⊂ undergraduate
{ M.S., M.A., Ph.D. } ⊂ graduate
{ undergraduate, graduate } ⊂ ANY (status)
{ biology, chemistry, computing, ..., physics } ⊂ science
{ literature , music, ..., painting } ⊂ art
{ science, art } ⊂ ANY (major)
{ 0.0 — 1.99 } ⊂ poor
{ 2.0 — 2.99 } ⊂ average
{ 3.0 — 3.49 } ⊂ good
{ 3.5 — 4.0 } ⊂ excellent
{ poor, average, good, excellent } ⊂ ANY (GPA)
Birth_place (*City* ⊂ *Province* ⊂ *Country*).
Birth_date (*Day* ⊂ *Month* ⊂ *Year*).

Figure 2: A concept hierarchy table of the database.

IDB rules are defined on top of EDB predicates. For example, *award_candidate* and *pre_requisite* are two IDB predicates defined in Fig. 3.

$$
\begin{aligned}
& award\_candidate(Name) \leftarrow \\
& \quad status(X) = graduate \wedge gpa(Name) \geq 3.75. && (2.1)\\
& award\_candidate(Name) \leftarrow \\
& \quad status(X) = undergraduate \wedge gpa(Name) \geq 3.5. && (2.2)\\
& pre\_requisite(Course, Pre\_requisite\_course) \leftarrow \\
& \quad pre\_requisite(Course,\ Pre\_requisite\_course). && (2.3)\\
& pre\_requisite(Course,\ Pre\_requisite\_course) \leftarrow \\
& \quad pre\_requisite(Course,\ Required\_course), \\
& \quad pre\_requisite(Required\_course,\ Pre\_requisite\_course). && (2.4)
\end{aligned}
$$

Figure 3: A set of deduction rules in IDB.

1. *All of the teaching assistants are graduate students.*

$$s \in Student \wedge c \in Course \wedge c.TA = s.Sname \rightarrow s.Status = graduate.$$

2. *Every teaching assistant has a good or excellent grade point average.*

$$s \in Student \wedge c \in Course \wedge c.TA = s.Sname \rightarrow s.GPA = \{good, excellent\}$$

Figure 4: A set of generalized rules stored in GDB.

Fig. 4 shows a set of generalized rules extracted from EDB and stored in GDB.

Our study below on intelligent query answering mechanisms will reference this database substantially. □

3 Four Categories of Query Answering Mechanisms in Knowledge-Rich Databases

As indicated in Section 1, queries in a knowledge-rich database can be classified into *data queries* and *knowledge queries*, and query answering mechanisms can be classified into *direct query answering* and *intelligent query answering*. Therefore, there are four basic combinations of queries and query answering mechanisms:

1. DD (Data query - Direct answering): direct answering of data queries;
2. DI (Data query - Intelligent answering): intelligent answering of data queries;
3. KD (Knowledge query - Direct answering): direct answering of knowledge queries; and
4. KI (Knowledge query - Intelligent answering): intelligent answering of knowledge queries.

In this section, query answering mechanisms are examined in each of these four categories.

3.1 Direct answering of data queries

Direct answering of data queries corresponds to direct data retrieval in knowledge-rich databases. Traditional query processing in relational and deductive databases belongs to direct answering of data queries. A *primitive-level data query* can be processed directly using relational and deductive query processing techniques.

A *high-level data query* can be processed in two steps. First, a query rewriting process can be performed to rewrite the query into one or a set of equivalent *primitive-level data queries* by substituting each high-level concept in the query with a set of or a range of its subordinate primitive-level concepts by consulting concept hierarchies in the KDB. Second, each rewritten query is then fed into a relational or deductive query processor for processing. Answers should be returned at the primitive level. Presentation of answers at a nonprimitive level, when desired, is considered as a task of *intelligent* query answering and will be discussed in the next subsection. One example is illustrated below.

Example 3.1 To *find the graduate students born in Canada, majoring in science, and with excellent GPAs*, the query can be formulated in a syntax similar to SQL as follows.

retrieve *Name*
from *Student*
where *Status* = *"graduate"* **and** *Major* = *"science"*
  and *Birth_place* = *"Canada"* **and** *GPA* = *"excellent"*

Notice that "*graduate*", "*science*" and "*excellent*" are high-level concepts which are not stored in the relation *Student*. Using the information stored in concept hierarchy H, the query can be reformulated by substituting *graduate* with {*M.S., M.A., Ph.D.*}, and *GPA* = *"excellent"* with $GPA \geq 3.5$ *and* $GPA \leq 4.0$, etc. The rewritten query can be answered by direct data retrieval. □

3.2 Intelligent answering of data queries

Intelligent answering of data queries refers to the mechanisms which answer data queries cooperatively and intelligently. Intelligent query answering is accomplished by analyzing the intent of a query and providing some generalized, neighborhood, or associated answers. There are many ways for a data query to be answered intelligently, including generalization and summarization of answers, explanation of answers or returning intensional answers, query rewriting using associated or neighborhood information, comparison of answers with those of similar queries, etc. Several mechanisms for intelligent answering of data queries using (generalized) database knowledge are examined.

3.2.1 Analysis of the intent of a query

To answer a query intelligently, the first important step is to analyze the intent of the query, determine whether it is necessary to provide assisted answers, and if it is, what kind of assistance should be provided. Such an analysis should be based on the available or discovered knowledge about database, queries, and users. Since a large volume of knowledge may exist or can be discovered in a database, one may often find that there exist too many "intelligent" ways to associate a query with the available or discoverable database knowledge. It is crucial to have knowledge about user's background and the role that he/she plays in order to understand user's intention, avoid superfluous answers, and provide users with quality assistance.

When posing a query, different users often have quite different intentions. For example, when asking the highest monthly balance of an account, a customer and a bank manager likely have different intentions. Therefore, an important task of query intent analysis is *user modeling*, which analyzes the user's background and intention and constructs different models for different classes of users.

Several interesting methods for query intent analysis have been developed in the studies on intelligent query answering [9, 10, 15, 4, 21]. Such analyses are based on the notions of generalization, association, aggregation, concept clustering, etc. Semantic data modeling, classification of *topics of interests*, and *plan analysis and formation* are powerful techniques for query intent analysis [10, 15, 4, 21], which can be applied to the analysis of query intent in the KDB.

Since the knowledge-rich database is constructed based on an extended (deductive) entity-relationship model together with the deductive and knowledge discovery components, the new components provide powerful support for query intent analysis. Considering an airline reservation system as an example, the method for query intent analysis in the KDB is outlined as follows.

1. **Data classification and concept clustering**: Based upon the extended entity-relationship model of the system, entities, relationships, attributes and specific conditions can be classified and clustered. For example, departure time and arrival time can be associated with time_table, airports can be clustered according to some local distance, etc. The data classification and clustering task is facilitated by the availability of concept hierarchies and knowledge discovery tools.

2. **User modeling**: Based upon user's profession/position (e.g., manager, clerk, business customer, tourist, etc.), confidence level (e.g., eligibility of accessing some sensitive data), accessing history (e.g., frequent flyer, business class traveler, being interested in some particular airlines, new customer, etc.) or other related information, a user can be associated with a particular user category built in the system. The linkage between a category of users and a class of preferred concepts or entity sets is constructed by experts in the development of intelligent query answering system. With the available knowledge-rich data model and knowledge discovery components, users can be naturally categorized into some high-level user classes (e.g., luxury, economy, or regular classes

for travelers) and be associated with a set of high-level concepts (e.g., the traveler's major interests expressed at a high concept level) to assist query intent analysis.

3. **Query classification**: A query can also be classified into different categories according to the query condition and the information to be inquired. For example, queries on travel plan can be categorized into long distance travel, short distance travel, etc. according to the conditions given in the query, or categorized into general browsing, detailed examination, ticket booking according to query actions. A query class can be linked with certain user categories, generalized concept classes and transformation rules to guide appropriate intelligent query answering for particular classes of queries.

4. **Transformation rule specification**: A set of transformation rules can be specified by experts based upon user category, query category, concept hierarchies and the relationships among high-level entities, attributes, and conditions. For example, if a user is in the category of tourist and new customer, the cost could be of a major concern at flight booking, and the information about low airfares could be of major interest. Such heuristics can be specified as transformation rules to guide intelligent query answering.

Query intent analysis can be performed by systematically applying techniques of user modeling, concept classification and clustering, query classification and transformation. Further, the constructed models and transformation rules should be testified by experiments and be tuned according to their effectiveness in intelligent query answering and the feedbacks from users. The query rewriting and answer transformation processes discussed below are directed by the results of query intent analysis.

3.2.2 Query rewriting using associated or neighborhood information

Direct data retrieval may not always find enough answers for a user. Furthermore, a user may like to know more information than the direct answers to a query for decision making. Therefore, it is often useful to provide associated or neighborhood answers to a query. Associated query answering can be performed by (1) *presenting the information about some additional attributes which are not directly inquired but are relevant to the query*; (2) *relaxation of certain query conditions*; and (3) *adding an alternative query which is closely related to the original one* [4].

Let the answer set be viewed as a relation table. Three mechanisms can find their corresponding relational transformations: *width-extension, height-extension*, and *table-extension.*

1. **Width-extension**: The first case (addition of relevant attributes) can be viewed as extension of the width of the answer table by adding some closely related attributes to the table. For example, an inquiry on the arrival time for air-flight booking can be answered by returning also the departure time and the possible transfer time, as well.

2. **Height-extension**: The second case (relaxation of certain query conditions) can be viewed as an extension of the height of the table. For example, an inquiry on the available flights for air-flight booking can be answered by relaxation of the maximum dollar restriction, flight-time restriction, airline selection restriction, etc.

3. **Table-extension**: The third case (answering an alternative query) can be viewed as an extension of the answer table or a switch to a similar table. For example, an inquiry on the available flights for air-flight booking can be answered by returning the flights to a neighborhood arrival or departure airports, or even a suggestion of other means of transportation, such as by train, ferry, or bus, depending on the distance, time and cost of the transportation, etc.

Query rewriting rewrites a query according to the intent of the query. Clearly, query intent analysis plays an important role in the selection of appropriate extensions. For example, the selection of associated additional attributes (as width-extension) should be determined by analysis of the semantics of the query and the associated attributes at the higher concept level, and the selection of relaxed constraints (as height-extension) should be based on the analysis of query semantics and the role of query constraints.

Query rewriting can be implemented by mapping query constants to an appropriate level via generalization or specialization and mapping a query to a neighborhood one by providing with additional, associated or neighborhood information. The knowledge discovery components, which specify or discover generalization, aggregation, neighborhood, or association relationships among data in the database, provide important assistance in the analysis of query intent and in the rewriting of queries into their alternatives based on hierarchical or neighborhood relationships.

Example 3.2 Consider a query to find the names of the teaching assistants of database courses in the University database. The query can be rewritten in the following ways.

1. *table width extension*: e.g., providing more information about the teaching assistants, such as their GPA, courses_taken, teaching experience, etc.

2. *table height extension*: e.g., providing teaching assistant information for other related courses, such as other computing science courses.

3. *table extension*: e.g., providing other relations, such as information about research assistants and project assistants in computing science if the user is a graduate student and the time is the beginning of a new semester (a job hunting season).

Obviously, the success of a query rewriting depends on the query intent analysis and the availability of associated, generalized and neighborhood information. Such information may exist in concept hierarchies or discovered knowledge rules or can be discovered by knowledge discovery tools. □

3.2.3 Answer transformation and answer explanation

Together with the rewriting of queries, the set of answers may also be transformed, explained, compared or summarized in different ways for intelligent query answering.

Generalization and summarization of answers

A database user may be interested in general description or overall statistics of the answer set to a query but not interested in the detailed answer set itself. Thus, a data query can be answered by generalization and summarization of the answer set, that is, by presenting generalized data only, a combination of generalized and primitive data, or a summarization of concrete answers (possibly together with the presentation of concrete answers) using generalized data and database statistics. Such a process belongs to **answer transformation**.

Example 3.3 A query which inquires about the information of a student Tom Jackson in the University database can be answered as "*Tom Jackson is an undergraduate student* (a concept at a level higher than *senior* student), *born in Canada* (not mentioning the specific city and province) *in 1971* (not mentioning the specific date)". This is meaningful if the user (such as a university administrator) is concerned of the general information but not the detailed one. Also, a query which inquires "*who have good or excellent GPAs in computing science*?" can be answered intelligently in several ways: (1) "*100% graduate students, 55% senior students, and 25% junior students*" (general, statistical information only), (2) "*all of the graduate students and the following undergraduate students ...*" (a combination of generalized and primitive data), (3) concrete answer (student names) plus a summarization of the answers at a high level, etc. □

By presentation of general information or associating such information with concrete answers, answers to a query can be presented in a general and concise manner, thus making the implications of the answers better understood.

An important technique for answer transformation is the *mapping between different levels of data based on concept hierarchies*. Constants in a query or answers to a query can be mapped up or down along a concept hierarchy depending on the semantics and the intent of the query. A high-level query can be rewritten into a primitive-level one by mapping the high-level data in the query to a set of primitive data using concept hierarchies. Similarly, a low-level answer set can be transformed into a high-level one by mapping a set of primitive data in the answer set to a set of corresponding high-level ones according to user's need. The interactions between query conditions and rule bodies (conditions) also need the data/constant mapping among different levels. For example, to examine whether a query is relevant to a certain generalized relation, the query can be restated at the same concept level as that in the rule.

Another important technique for answer transformation uses *lazy evaluation*, that is, providing rule bodies (conditions) without presenting the full answers set. Detailed and concrete answers are provided only by further requests. Lazy evaluation as an intelligent query answering mechanism has been studied in deductive database research [10, 16, 21, 20]. Instead of returning the concrete answer set, the query answering mechanism instantiates a deduction rule using query constants and returns the instantiated condition (body) of the rule or a mixture of instantiated rule condition (body) and concrete data as the answers to the query. Assume that the generalized rule is also in the form of "head $\leftarrow$ body". Lazy evaluation can be performed by returning the body of a generalized rule if the query matches the head of the rule. Besides directly using the available rules, generalized rules can also be obtained by further generalization on the portion of an intermediate generalized relation which matches the query conditions.

Furthermore, generalization and summarization of answers can be implemented by taking advantage of the available generalized information and knowledge discovery tools. If there is a corresponding generalized rule, the processor returns the instantiated body of the rule when the query matches the head of the rule. If there is a corresponding intermediate generalized relation, further generalization and summarization can be performed on the portion of intermediate generalized data which matches the query condition. Further generalization may produce a generalized rule with a summary of statistical information in terms of generalized concepts. Otherwise, when there is no corresponding generalized information, generalization is performed by first retrieving the required answer set and then performing generalization on the retrieved answer set using the knowledge discovery tools.

Answer explanation

Another intelligent query answering method is **answer explanation**, which explains the answers to a query by presentation of the associated rules, demonstration of the reasoning process, or illustration of the general information. The summarization of the statistics of an answer set discussed above can also be employed as a technique for answer explanation.

The following example demonstrates that it is often necessary to provide explanations to the answers when the query condition follows or contradicts a rule or an integrity constraint.

Example 3.4 If a query condition follows or contradicts a rule or an integrity constraint, the query can be answered by presentation of the knowledge (such as the rule) rather than primitive data. Data retrieval is necessary only if the direct presentation of primitive data is explicitly required. For example, suppose there is a generalized rule, "*all of the teaching assistants are graduate students*". The query "*find all of the undergraduate students who are teaching assistants*" can be answered by returning an empty set without accessing extensional database. However, it is user-friendly to also give an explanation by simply presenting the rule itself. Similarly, if "*all of the teaching assistants have good or excellent GPAs*" is a generalized rule, the query "*find all of the teaching assistants whose GPAs are greater than 2.5*", may return "*all of the teaching assistants*", together with the rule. Specific teaching assistant names are presented only when the user requests for more details. □

The process described above can be implemented by testing of the query condition against the rule for containment or contradiction. If the query passes the test, lazy evaluation can be applied rather than returning detailed answers.

Answer comparison

Queries can also be answered intelligently by **answer comparison**, which compares or contrasts the general characteristics of its answers with some similar queries. Answer comparison may involve two steps: (i) rewriting a query into a neighborhood query, and (ii) generalization, summarization and comparison of two answer sets, one to the original query and one to the neighborhood query, at a general level. The first step, rewriting a query into a neighborhood query, can be performed by query intent analysis and substitution of some query constant(s) in the original query by some similar concept(s) using the knowledge about concept hierarchies. The second step involves learning characteristic and discriminant rules using knowledge discovery techniques [6] which will be explained further in Section 4.

Example 3.5 In answering the query, "*find all of the graduate students with excellent GPAs*", it is interesting to find the undergraduate students with similar characteristics or the graduate students with weaker GPAs and compare the general characteristics and statistics between these answers. Such comparisons may lead to some interesting observations, such as "*more than 50% graduate students but only about 20% undergraduate students have excellent GPAs*". □

3.3 Direct answering of knowledge queries

A knowledge query is a statement which inquires about database knowledge, including concept hierarchies, deduction rules, integrity constraints and general characteristics of a particular set of data in a database. Direct answering of knowledge queries means that a query processor receives a knowledge query and answers it directly by returning the inquired knowledge. Since IDB knowledge and concept hierarchy information are stored in the database according to our assumption, a query on such knowledge can be answered by direct retrieval. However, the situation is different at querying generalized knowledge. A generalized database (GDB) usually stores only a small, but frequently used portion of generalized knowledge. Thus, an inquiry on general knowledge should be answered by direct retrieval only if the knowledge is available in GDB. Otherwise, the knowledge should be discovered dynamically by a knowledge discovery process, which will be discussed in detail in Section 4. In general, a knowledge query can be answered by consulting concept hierarchy, retrieving stored rules (if available) or triggering a discovery process.

Different syntactic specifications can be adopted to distinguish knowledge queries from data queries. A data query is to *retrieve the data elements that satisfy a condition* Φ; whereas a knowledge query is to *describe the data elements that satisfy* Φ. Following the notion proposed by Motro and Yuan [16], data queries and knowledge queries are distinguished in syntax by starting the former with **retrieve** but the latter with **describe**. Further, to distinguish different types of knowledge being inquired, concrete keywords such as *generalized rule, deduction rule, concept hierarchy, integrity constraint*, etc. can be used after the keyword **describe**. Moreover, to query a discriminant rule which distinguishes the general characteristics of one class (*target class*) from others (*contrasting classes*), the following syntax is adopted:

> **describe generalized rule for** *relation*
> **which distinguishes** *target_class* **from** *contrasting_class*
> **where** *condition* Φ.

Several knowledge queries are presented in the following examples.

Example 3.6 To find the deduction rule *award_candidate* for Canadian graduate students, a query can be formulated as below.

> **describe deduction rule** *award_candidate(candidate)*
> **where** *Status(candidate)* = *"graduate"* **and** *Birth_place(candidate)* = *"Canada"*

This query can be answered by direct retrieval of deduction rules. Notice that only the condition, *Status(candidate)* = *"graduate"*, matches the boody of a deduction rule for *award_candidate*, which indicates that there is no further distinction on birth place in the condition for an award candidate. Thus the rule (1a) is presented as the answer to the query. □

Example 3.7 To *describe the characteristics of the graduate students in computing science who were born in Canada with excellent GPA*, the query can be formulated as below.

> **describe generalized rule**
> **for** *Student*
> **where** *Status* = *"graduate"* **and** *Major* = *"cs"*
> **and** *Birth_place* = *"Canada"* **and** *GPA* = *"excellent"*

Notice that the query represents a high-level knowledge query since "*graduate*", "*Canada*" and "*excellent*" are not stored as primitive data in the *University* database. The query can be answered by directly retrieving the discovered rule, if available, or by performing induction on the relevant data set [6]. □

Example 3.8 To *distinguish the characteristics of the graduate students from undergraduate students in computing science, born in Canada with excellent GPA*, the query can be formulated as below.

> **describe generalized rule for** *Student*
> **which distinguishes** *Status* = *"graduate"*
> **from** *Status* = *"undergraduate"*
> **where** *Major* = *"cs"* **and** *Birth_place* = *"Canada"* **and** *GPA* = *"excellent"*

Notice that the query wishes to find a discriminant rule which contrasts the general properties of the two classes. The rule can be discovered dynamically by a knowledge discovery process from primitive data or from an intermediate generalized relation as illustrated in Section 4. □

3.4 Intelligent answering of knowledge queries

Intelligent answering of knowledge queries means that a knowledge query is answered in an intelligent way by analyzing the intent of the query and providing generalized, neighborhood or associated information. Similar to the intelligent answering of data queries, a knowledge query can be answered in many ways, such as generalization and summarization of answers, explanation of answers, query rewriting using associated or neighborhood information, comparison of answers with those of neighborhood queries, etc. The availability of database knowledge and knowledge discovery tools enhances the power and efficiency of intelligent query answering of knowledge queries. The ideas are illustrated in the following examples.

Example 3.9 The knowledge query of Example 3.6, which is to find the deduction rule *award_candidate*, can be answered intelligently not only by returning the *award_candidate* rule eligible to Canadian graduate students but also by (i) providing an explanation that both Canadian and foreign graduate students share the same condition for the award, (ii) returning the *award_candidate* rule eligible for undergraduate students as well, or (iii) returning other associated information, such as award name, amount, application deadlines, regulations, summary of award history, or statistical information, etc. □

Example 3.10 The knowledge query of Example 3.7, which is to find the characteristics of designated graduate students, can be answered intelligently by returning the characteristic rule for Canadian graduate students with excellent GPA's, together with (i) the characteristics of Canadian graduate students with different majors or weaker GPAs for comparison, or (ii) an explanation of the reasons why such students got excellent GPA's. □

Intelligent answering of knowledge queries can involve great complexity in query intent analysis and demand sophisticated implementation techniques. Therefore, the efficient realization of the underlying mechanisms is an interesting issue for future research.

4 Knowledge Discovery and Its Role in Intelligent Query Answering

Discussions in the previous section focused on a wide spectrum of possible data and knowledge queries which demand implementation techniques ranging from well-studied data retrieval methods to newly developed knowledge discovery techniques. Knowledge discovery plays an important role in the realization of intelligent query answering. In this section, detailed discussion is dedicated to the knowledge discovery techniques and their roles in intelligent query answering.

4.1 Knowledge discovery tools

Since both direct answering of a knowledge query or intelligent answering of a data or knowledge query may require data generalization, statistics summarization, and knowledge rule extraction, it is important to provide efficient knowledge discovery tools for query processing.

Generalization can be classified into two categories: (1) *simple attribute generalization*, and (2) *generalized rule extraction*. The former refers to data expressed in the answer set using high level concepts, such as the answer, "*Tom Jackson is an undergraduate student, born in Canada in 1971*", as shown in Example 3.3. Such attribute generalization can be performed by simply substituting some low level data in the answer set by the corresponding superordinate concepts at an appropriate level based on the query intent analysis. *Generalized rule extraction* requires a relatively sophisticated knowledge discovery process. Together with many other knowledge discovery techniques [5, 11, 18], an attribute-oriented induction method has been developed and tested in Simon Fraser University [1, 6]. The method is modified to facilitate the intelligent query answering process. The major modification, rather than deriving a generalized rule, is to produce a special intermediate generalized relation, *prime relation*, which facilitates the extraction of different feature tables and the generation of variant generalized rules for different purposes of intelligent query answering. The derivation of the prime relation is discussed in this subsection and the extraction of feature tables is studied in the next subsection.

It is desirable to express a generalized result using a small number (usually no more than 8) of distinct (generalized) values for each attribute in the generalized relation. Sometimes a user or an expert may like to specify explicitly a designated concept level as the desirable level for an attribute, such as the level "country" for a birth_place, etc. Otherwise, a small integer is usually specified (or taken by default) as a **desirability threshold** for an attribute. An attribute is at the *desirable* level if it contains no more distinct values than its *desirability threshold*. It is **generalizable** if the attribute contains a relatively large number of distinct values in the relation but there exists a concept hierarchy for the attribute (i.e., there are higher level concepts which subsume these attribute values). Otherwise, it is **nongeneralizable**.

Attribute-oriented induction is performed in three steps. At the first step, a set of task-relevant data is collected by execution of a (direct) data query. At the second step, the collected data is generalized by (1) removal of nongeneralizable attributes, and (2) performing concept-tree ascension (replacing lower-level attribute values in a relation by their corresponding superordinate values using the concept hierarchy) on each generalizable attribute to its desirable level. By removal of nongeneralizable attributes and generalization of the values in the generalizable attributes to the desirable level, some generalized tuples in the relation may become identical. The identical generalized tuples are merged into one tuple, associated with a special internal attribute, *vote*, which registers the number of original tuples generalized to the current one. The generalized relation obtained at this stage is called the **prime relation**. The third step can actually be performed in two different directions. One direction is to derive a final generalized relation by continuing further generalization by attribute-oriented induction. The other direction is to derive a generalized feature table for intelligent query answering, which is presented in the next subsection. The derivation of a final generalized relation is performed by further generalization on the attributes in the prime relation based on preferred rule forms, tuple reduction ratio, preferred concept levels of the attributes, etc. and then merge of identical generalized tuples with vote accumulation. The final generalized relation, which consists of only a small number (usually controlled by a *generalization threshold*) of distinct generalized tuples, can be expressed in a logic form with the association of statistical information.

The core of the attribute-oriented induction is the concept-tree ascension on generalizable attributes, which relies heavily on the concept hierarchy (H) information available in the KDB. The concept hierarchy for numerical values can be constructed dynamically without prior knowledge based on the value range distribution in the database. For other hierarchies found in the KDB, modification can be performed automatically based on the statistics of relevant data sets and user preference in order to extract interesting rules. For example, to study the relationships between GPA and $Birth\_place$, it is wise to modify a given hierarchy to allow detailed classification of the students who were born in a local province or a local country.

A prime relation maintains the relationships among generalized data in different attributes for a frequently inquired data set at a reasonable cost. The prime relation can be used for extraction of various kinds of generalized rules. Therefore, the prime relations for frequently inquired data sets should be stored in the GDB to facilitate intelligent query answering. The algorithm for the extraction of the prime relation is presented as follows.

Algorithm 4.1 *Extraction of the prime relation from a frequently inquired data set R.*

Input: (i) A frequently inquired data set R, a relation of arity n with a set of attributes A_i $(1 \leq i \leq n)$; (ii) a set of concept hierarchies, H_i, where H_i is a hierarchy on the generalizable attribute A_i, if available; and (iii) a set of *desirability thresholds* T_i for each attribute A_i.

Output. The prime relation R'.

Method.

> $R' := R$; /* R represents a frequently inquired data set, obtained by a data query. */
> **for each** attribute A_i in R' **do** {
>   **if** A_i is nongeneralizable **then** remove A_i;
>   **if** A_i is not at the desirable level but generalizable
>   **then** generalize A_i to the desirable level;
> }
> Merge identical tuples in R' (with the number of identical tuples registered in *vote*).
> return (R').

Note: Generalization is implemented as follows. First, collect the distinct values in the relation and compute the lowest desirable level L on which the number of distinct values will be no more than the desirability threshold T_i by synchronously ascending the concept hierarchy from these values. Then generalize the attribute to this level L by substituting for each value of A_i's with its corresponding superordinate concept in Hi at level L [6]. □

Theorem 4.1 *The relation R' obtained according to Algorithm 4.1 is the prime relation of a data relation R.*

Proof. An attribute-value pair represents a conjunct in the logical form of a tuple. The removal of a conjunct eliminates a constraint and thus generalizes the rule. This corresponds to the generalization rule *dropping conditions* in *learning-from-examples* [14]. Thus, if an attribute is nongeneralizable, its removal generalizes the relation. Furthermore, if an attribute is not at the desirable level but generalizable, the substitution of an attribute value by its higher level concept makes the tuple cover more cases than the original one and thus generalizes the tuple. This corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [14]. Since all of the generalizable attributes are at the desirable level, the generalized relation is the prime relation. □

The attribute-oriented induction technique avoids the combinatorial explosion problem of tuple-oriented generalization, explores the integration of well-implemented relational operations with the learning-from-examples algorithm, and leads to an easily implementable and highly efficient algorithm for induction in large databases [6].

4.2 Extraction of generalized rules from prime relations

The derivation of generalized rules from prime relations includes the extraction of a generalized feature table from the prime relation and the generation of a variety of interesting generalized rules from the feature table.

A generalized feature table is a two-dimensional table generated directly from the prime relation. It represents the occurrence frequency of a set of generalized features in relevance to one or a set of reference attributes in the

Status	Sex	Age	Birth_country	GPA	IsTA	vote
grad	M	25_30	Canada	good	Y	8
grad	F	25_30	Canada	excellent	Y	2
...	...	...	...	...	...	...
underg	M	16_25	Asia	good	N	6

Table 1: A prime relation for the interested set of students.

Status	Sex		Age			Birth_country				GPA				IsTA		vote
	M	F	16_25	26_30	>30	Canada	USA	Asia	Europe	poor	avg	good	exclnt	Y	N	
grad	40	10	10	20	20	30	5	10	5	1	1	30	18	30	20	50
underg	120	80	140	60	0	130	40	30	0	15	90	70	25	0	200	200
total	160	90	150	80	20	160	45	40	5	16	91	100	43	30	220	250

Table 2: A *Status* feature table mapped from the prime relation.

prime relation. One such feature table is presented in the following example. Notice that the feature table presented here is in relevance to only one reference attribute, *Status*, however, the method can be easily extended to generate feature tables in relevance to more than one reference attribute.

Example 4.1 Suppose a frequently inquired data set in the University database collects the information about computing science students and their associated teaching assistant information. Let the prime relation be generalized from a data set obtained by a selection from the relation *Student* using, $Major$ = "*computing_science*", plus an attribute $IsTA$ to indicate whether the student is a teaching assistant (i.e., whether the tuple is joinable with the $Course$ relation on the teaching assistant attribute). The scheme for the initial data relation is,

$$Student(Sname, Status, Sex, Birth\_date(Day, Month, Year),$$

$$Birth\_place(City,\ Province,\ Country),\ GPA,\ IsTA).$$

Following Algorithm 4.1, the attributes *Sname*, *Day*, *Month*, *City*, and *Province* are removed, and the prime relation has the scheme,

$$Student(Status, Sex, Age, Birth\_country, GPA, IsTA),$$

where *Age* is from *Birth_date(Year)*, and *Birth_country* from *Birth_place(Country)*. After generalization, the prime relation contains only a small number of distinct values in each attribute as follows: *Status*: {*grad, undergrad*}, *Sex*: {*male, female*}, *Age*: {16_25, 26_30, >30}, *Birth_Country*: {*Canada, USA, Asia, Europe*}, *GPA*: {*poor, average, good, excellent*}, and *IsTA*: {*Yes, No*}. The prime relation is shown in Table 1.

Different feature tables can be extracted from the prime relation based on the interest of different attributes. For example, to compare students who possess different status (graduates vs. undergraduates), the prime relation can be mapped into a *Status* feature table (Table 2).

The *Status* feature table consists of 3 rows: each of the two distinct *Status* values in the prime relation {"*grad*", "*undergrad*"} corresponds to one row, and the last row (*total*) is the summation of information in the previous rows. The feature table consists of 5 major columns, each corresponding to one attribute in the prime relation, plus one special column for *vote*. Each major column in the table is further divided into k subcolumns (where k is the number of distinct general values in the corresponding major column), each corresponding to one distinct value in the attribute. For example, *GPA* is divided into 4 subcolumns: {*poor, avg, good, exclnt*}, each corresponding to one distinct value in *GPA*.

The table contents are derived from the prime relation as follows. Each slot in the table (except for the last row) corresponds to the number of occurrences of the corresponding values in the prime relation. For example, the slot for "*grad*" and "*good (GPA)*" corresponds to the number of grad's with good GPAs, that is, the summation of all of the votes of those rows with *Status* = "*grad*" and *GPA* = "*good*" in the prime relation. The special column *vote* registers the number of occurrences of the corresponding class in the relation. For example, 50 in "*grad*" indicates that there are 50 graduates in total in the prime relation. The special row *total* summarizes the total number of occurrences with each feature in all of the classes. For example, total = 160 in the column "*Sex = M*" indicates that there are totally 160 male students computed in the prime relation. □

The extracted feature table is useful for derivation of the relationships between the classification attribute and other attributes at a high level. For example, the generalized rule, *all of the teaching assistants are graduate students*, can be extracted from Table 2 based on the fact that the class *grad* takes all of the *IsTA* count. The feature table is especially useful for generation of rules associated with quantitative (statistical) information. For example, the following quantitative rule can be derived from the first row *grad* and the first major column *Sex*,

$$grad(x) \rightarrow male(x)[80\%] \vee female(x)[20\%].$$

The rule indicates that *80% of (computing_science) graduate students are male and 20% are female.* Obviously, such a feature table is useful for presenting neighborhood and summary information in intelligent query answering.

In general, the following algorithm extracts a feature table from a prime relation.

Algorithm 4.2 *Extraction of the feature table T_A for an attribute A from the prime relation R'.*

Input: A prime relation R' consists of (i) an attribute A with distinct values $a_1, \ldots, a_m$, (ii) k other attributes $B_1, \ldots, B_k$ (suppose different attributes have unique distinct values), and (iii) a special attribute, *vote*.

Output. The feature table T_A for the attribute A.

Method.

1. The feature table T_A consists of $m + 1$ rows and $l + 1$ columns, where l is the total number of distinct values in all of the k attributes. Each slot of the table is initialized to 0.
2. Each slot in T_A (except the last row) is filled by the following procedure,

```
for each row r in R' do {
        for each attribute B_i in R' do
                T_A[r.A, r.B_i] := T_A[r.A, r.B_i] + r.vote;
        T_A[r.A, vote] := T_A[r.A, vote] + r.vote; }
```

3. The last row p in T_A is filled by the following procedure:

```
for each column s in T_A do {
        for each row t (except the last row p) in T_A do
                T_A[p, s] := T_A[p, s] + T_A[t, s];
}
```

□

Theorem 4.2 *Algorithm 4.2 correctly registers the number of occurrences for each general feature in the prime relation R'.*

Proof. Following the algorithm, each tuple in the prime relation is examined exactly once with every feature registered in the corresponding slot in the feature table. Their column-wise summation is registered in the last row. □

Since a feature table registers the number of occurrences for each general feature in the prime relation, the quantitative relationships between the corresponding general features can be easily referenced and transformed into a set of interesting general rules to facilitate intelligent query answering. The following example shows how a generalized feature table may facilitate such query answering.

Example 4.2 Let Table 2 be the *Status* feature table extracted from the prime relation. Suppose the query is to find a generalized rule which *distinguishes graduate and undergraduate CS students born in Canada with excellent GPA* (Example 3.8). This requires a discriminant rule which, unfortunately, cannot be directly extracted from the feature table above since the feature table does not preserve all of the information contained in the prime relation. For example, the information for graduate and undergraduate students in relevance to both *Birth_country = "Canada"* and *GPA = "excellent"* cannot be inferred from the generalized feature table. The required discriminant rule can be extracted from the prime relation by selecting those rows with both *Birth_country = "Canada" and GPA = "excellent"*, partitioning them according to their status, further generalizing them when necessary, and then contrasting their features [6]. However, the feature table provides assistance to intelligent query answering: intelligent answers to the above query can be extracted from the feature table because the feature table is a column-wise summarization of quantitative information stored in the prime relation. For example, the relationships between *Status* and *Birth_country*, *GPA*, etc. can be directly extracted from the feature table with the following results, Among 50 CS graduate students, 60% born in Canada, 10% in Asia, 5% in the U.S. and 5% in Europe; moreover, among these students, 36% have excellent GPAs, 60% have good GPAs, and 4% have average or poor GPAs, etc. In contrast, among 200 undergraduate students, 65% born in Canada, 20% in USA, ...

Furthermore, comparison of two classes can be based on the column-wise information in the feature table, such as,

Among 43 CS students with excellent GPAs, 41.86% are graduates and 58.14% are undergraduates; and among 160 CS students born in Canada, 18.75% are graduates and 81.25% are undergraduates, etc. □

The following algorithm extracts generalized rules from a feature table.

Algorithm 4.3 *Extraction of generalized rules from the feature table T_A.*

Input: A feature table T_A for the attribute A, where A has a set of distinct generalized values $a_1, \ldots, a_m$. Another attribute B in the table has a set of distinct generalized values $b_1, \ldots, b_n$. The slot of the table corresponding to the row with the value a_i and the column with the value b_j is referenced by $T_A[a_i, b_j]$.

Output. A set of generalized rules relevant to A and B extracted from the feature table.

Method.

1. For each row a_i, the following rule is generated in relevance to attribute B, which presents the distribution of different generalized values of B in class a_i.

$$a_i(x) \rightarrow b_1[p_{i_1}] \vee \ldots \vee b_n[p_{i_n}].$$

where p_{i_j} is the probability that the value b_j of B is in class a_i, which is computed by,

$$p_{i_j} = T_A[a_i, b_j]/T_A[a_i, vote].$$

2. For each column b_j, the following rule is generated in relevance to all of the classes, which presents the distribution of the generalized value b_j of B among all of the classes.

$$b_j(x) \rightarrow a_1[q_{1_j}] \vee \ldots \vee a_m[q_{m_j}].$$

where q_{i_j} is the probability that the value b_j of B is distributed in class a_i among all of the classes, which is computed by,

$$q_{i_j} = T_A[a_i, b_j]/T_A[total, b_j].$$

□

Theorem 4.3 *Algorithm 4.3 correctly generates relationships between two attributes A and B at a high level using the feature table.*

Proof. Theorem 4.2 shows that the feature table registers all of the number of occurrences of each feature and two specific properties, *total* and *vote*, in each class. The above algorithm extracts generalized rules from each column and each row in the feature table by computing the proportion of the number of occurrences of each generalized feature vs. its corresponding total number of occurrences in each row or each column. Thus, we have the theorem. □

By extraction of generalized feature tables and generation of different generalized rules, many of the intelligent query answering mechanisms which require knowledge discovery tools discussed in the previous section can be implemented efficiently. For example, summarization of statistics of the answer set, comparison of the answer set with those of neighborhood queries, etc. can be realized by extraction of different rules from the corresponding generalized feature table.

4.3 Semantic query optimization using generalized knowledge

Semantic query optimization method applies database semantics, integrity constraints and knowledge rules to query optimization [2]. Techniques have been developed for semantic query optimization based on database semantics, deduction rules and integrity constraints [2, 12, 13]. With the availability of concept hierarchies and generalized knowledge, new techniques can be explored to enhance the power and applicability of semantic query optimization.

A generalized rule can be treated like a deduction rule or an integrity constraint in semantic query optimization. Therefore, the techniques developed for semantic query optimization in relational and deductive databases [2, 12, 13] can be directly applied to knowledge-rich databases with discovered knowledge. Furthermore, with the availability of concept hierarchies and generalized knowledge, query optimization using a generalized rule can be explored for the query conditions which are subsumed by the rule body (condition) or the rule head (conclusion) at a different level of concept hierarchies. This is based on the following theorem.

Theorem 4.4 *(*Condition specialization and conclusion generalization*) Let a knowledge rule be in the form of*

$$p_1 \wedge p_2 \wedge \ldots \wedge p_i \wedge \ldots \wedge p_n \longrightarrow q.$$

The specialization of a condition pi to pi′ and/or the generalization of the conclusion q to q′ based on that the information in concept hierarchies will not change the validity of the rule. That is,

$$p_1 \wedge p_2 \wedge \ldots \wedge p_i' \wedge \ldots \wedge p_n \longrightarrow q.$$

$$p_1 \wedge p_2 \wedge \ldots \wedge p_i \wedge \ldots \wedge p_n \longrightarrow q'.$$

Proof. Since p_i' is a specialization of p_i, $p_i' \longrightarrow p_i$. Similarly, $q \longrightarrow q'$. Based on the transitivity property of logic rules, the above rules hold. □

As a simple example for the theorem, if *the GPA of every graduate student is greater than 3.2*, then *the GPA of every M.Sc. student must be either good or excellent.*

According to Theorem 4.4 and the principles of semantic query optimization in relational and deductive databases [2, 12, 13], the semantic query optimization techniques for application of generalized knowledge are presented as follows.

1. **Query condition subsumed by the rule body (condition)**: *If a query condition is subsumed by the body of a rule, the conclusion (head) of the rule applies.*

   For example, if there is a generalized rule: *all of the teaching assistants for 400 level courses are Ph.D. students*, then querying the status of a student who is assisting 459 will return *"Status = Ph.D."* without searching the EDB. Further, the rule can be returned as an explanation to the answer.

2. **Query condition conflicting with the rule conclusion**: *If some query conjunct(s) in a conjunctive query is subsumed by the body of a rule but some other conjunct(s) conflicts with the head (conclusion) of the rule, the answer to the query is empty.*

   For example, if there is a generalized rule: *all of the teaching assistants are graduates*, then querying the courses which an undergraduate is assisting returns an empty set without searching EDB. Also, the rule can be presented as an explanation.

3. **Query conjunct elimination**: *If a query conjunct is implied by another condition in the query, the query conjunct can be removed from the query.*

   For example, if a query contains both *S.Status = "M.Sc."* and *S.GPA ¿ 3.0*, the second conjunct can be removed from the query since it is implied from the first one in the generalized rule.

4. **Query conjunct introduction**: *If a query conjunct is subsumed by the body of a rule, the head (conclusion) of the rule can be introduced as a new conjunct to the query if such an introduction may improve search efficiency (e.g. exploration of indexing or clustering properties of databases, etc.).*

   For example, to *find all of the foreign students majoring in science with GPA between 3.2 to 3.4*, the query is subsumed by the body (condition) of a generalized rule in GDB: *all of the foreign students majoring in science with a good GPA are graduate students.* The new conjunct, *Status = "graduate"*, can be added to the query. The search can be improved if the tuples in *Student* are grouped or partitioned according to *Status.*

5. **Sharpening query condition**. *If some query conjunct(s) is subsumed by the body (condition) of a rule, and the rule head (conclusion) introduces a more selective condition than some query conjunct, then the less selective query conjunct should be replaced by the more selective one.*

   For example, if some query condition is subsumed by a generalized rule which concludes that S.GPA = *"excellent"* ($\geq$ 3.5), then a less selective query conjunct, *S.GPA* $\geq$ 2.0, can be replaced by a more selective one.

6. **Query in relevance only to generalized knowledge**. *If the conditions of a query and its inquired information are relevant only to the generalized knowledge, it can be answered by consulting the generalized knowledge only.*

   For example, a query, "*how many graduate students in Computing Sciences were born in foreign countries?*", can be answered directly by examining the corresponding prime relation if it is stored in the GDB.

As a summary of the discussion, an algorithm is presented here, which explores semantic query optimization using generalized rules.

Algorithm 4.4 *Semantic query optimization using generalized rules.*

Input: (i) A set of generalized rules R, (ii) a set of concept hierarchies, H, and (iii) an input data query q which consists of a set of conjuncts.

Output. A possibly optimized processing plan for query q.

Method.

1. Test whether there is a conjunct ci implied by another conjunct cj in the conjunctive query q. If there is, remove ci.
2. Test whether some query conjunct(s) in a conjunctive query is subsumed by the body (condition) of a rule but some other conjunct(s) conflicts with the head (conclusion) of the rule. If it is so, the answer set is empty.
3. Test whether all of the (remaining) query conjuncts are subsumed by the body (condition) of a generalized rule. If so, return the conclusion (head) of the rule as the (intensional) answer to the query.
4. Test whether some of the query conjuncts are subsumed by the body of a generalized rule. If so, examine whether the conclusion (head) of the rule may (i) sharpen a conjunct in the query or (ii) reduce the search effort. If so, replace the conjunct by the rule head (conclusion) in case (i), and add the rule head (conclusion) as a new conjunct to the query in case (ii). □

Theorem 4.5 *Algorithm 4.4 correctly performs semantic query optimization using the rules stored in GDB.*

Proof. Step 1 corresponds to *query conjunct elimination*, Step 2 to *query condition conflicting with rule head (conclusion)*, Step 3 to *query conclusion subsumed by rule body (condition)* and *query in relevance only to generalized knowledge*, and Step 4 to *sharpening query condition* and *query conjunct introduction*. Thus we have the theorem. □

5 Discussion and Conclusion

A general framework has been presented for intelligent query answering in a knowledge-rich database composed of deductive and knowledge discovery components. The availability of generalized rules, concept hierarchies and knowledge discovery tools greatly enhance the power of intelligent query answering. First, it expands the spectrum of knowledge queries from inquiring deduction rules to inquiring general regularity of data, such as characteristic rules, discriminant rules, data evolution regularities, etc. [6]. Second, it facilitates the query intent analysis since the notions of generalization, aggregation, neighborhood, similarity, etc. can be studied systematically using the generalized knowledge and concept hierarchies. Third, it facilitates intelligent query answering since answers can be presented in general terms, summarized by statistical information, and compared with similar groups of data at a high level. Finally, the intelligent query answering can be implemented efficiently using generalized rules and knowledge discovery tools using prime relations, feature tables, semantic query optimization and other implementation techniques.

The enhanced power of intelligent query answering may lead to two problems: *superfluous "intelligent" answers* and the *risk on database security.*

The first problem indicates that one may suffer from obtaining too many superfluous "interesting" answers to a query because there are many ways for a query to be answered intelligently. Techniques should be developed to control the answer generation process in intelligent query answering. In general, one may assume that an appropriate knowledge level is associated with each user. A user usually poses queries at his/her corresponding knowledge level and expects the answers to be presented at the same level. If the contents of the answer set are not at such level, generalization or specialization should be performed on the answer set as concept level adjustment. Further, with user modeling and query intent analysis, only those answers which match the query intent and the user model will be presented. More desirably, an intelligent query answering process can be triggered or directed by interaction with users. For example, after obtaining the preliminary set of answers to a query, some following-up questions can be raised by users, such as "*in more detail?*", "*in summary?*", "*why?*", "*other options?*", "*comparing with others?*", etc. These questions indicate what kind of intelligent answers are expected. Then the corresponding intelligent query-answering mechanisms can be evoked.

The second problem indicates that with the extended power of intelligent query answering, some sensitive or confidential information could be disclosed inappropriately to someone who should not know it [17]. One technique which may enhance the database security is to associate with a user model certain kinds of constraints on accessing rights. For example, if the user is a student (easily known from the login name), the constraints on intelligent answering of his/her query in a university database will be quite different from the same query posed by a professor. Sensitive information will not be disclosed to the users who do not have appropriate access rights. However, because of the power and complexity of deduction and knowledge discovery, it is difficult to tell to what extent that accessing certain piece of information may eventually lead to the disclosure of sensitive information by a sequence of deduction and induction. Therefore, more study should be performed on ensuring database security in intelligent query answering in databases augmented with deduction and knowledge discovery components.

Based upon our previous study on knowledge discovery in databases [1, 6], a prototyped experimental database learning system, **DBLEARN**, has been constructed. The system is implemented in C with the assistance of UNIX software packages LEX and YACC (for compiling the DBLEARN language interface) and operates in conjunction with the SyBase DBMS software. Based on the principles of attribute-oriented induction, some experimentation has been performed for knowledge discovery in a relatively large database, a **Grant Information System** for *Natural Sciences and Engineering Research Council of Canada* (NSERC). Our experimental results on direct answering of data and knowledge queries are successful with satisfactory performance. Further studies are being performed on intelligent query answering using the provided concept hierarchies and knowledge discovery tools.

The innovations of this paper can be summarized as follows. First, a knowledge-rich data model is constructed which consists of an extended entity-relationship schema, an extensional database, an intensional database, a set of concept hierarchies, a set of generalized rules, and a set of knowledge discovery tools. Second, a systematic study has been performed on intelligent query answering with the focus on the application of discovered knowledge, concept hierarchies, and knowledge discovery tools to intelligent query answering in database systems. Query answering mechanisms are classified into (1) direct answering of data queries, (2) intelligent answering of data queries, (3) direct answering of knowledge queries, and (4) intelligent answering of knowledge queries. Third, techniques have been developed for implementation of such mechanisms using discovered knowledge and/or knowledge discovery tools, which include deduction, generalization, data summarization, rule discovery, concept clustering, query rewriting, lazy evaluation, semantic query optimization, etc. Some implementation techniques for knowledge discovery and semantic query optimization are investigated in detail.

Our study shows that knowledge discovery substantially broadens the spectrum of intelligent query answering and may have deep implications on query processing in data- and knowledge-base systems. There are many interesting issues which should be studied further. The systematic development of language primitives for intelligent query answering, new techniques for implementation of intelligent query answering mechanisms using knowledge discovery techniques, mechanisms for intelligent answering of knowledge queries, security problems in intelligent query answering, and the integration of different intelligent query answering techniques are interesting topics for future research.

Acknowledgements

The authors are grateful to the anonymous referees for their many valuable comments and suggestions that improved the quality of this paper.

References

[1] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.

[2] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15:162–207, 1990.

[3] P. Chen. The entity-relationship model : Toward a unified view of data. *ACM Trans. Database Syst.*, 1:9–36, 1976.

[4] F. Cuppens and R. Demolombe. Cooperative answering: A methodology to provide intelligent access to databases. In *Proc. 2nd Int. Conf. Expert Database Systems*, pages 621–643, Fairfax, VA, April 1988.

[5] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.

[6] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 547–559, Vancouver, Canada, August 1992.

[7] J. Han and Z. N. Li. Deductive-ER: Deductive entity-relationship model and its data language. *Information and Software Technology*, 34:192–204, 1992.

[8] X. Hu. Conceptual clustering and concept hierarchies in knowledge discovery. In *M.Sc. Thesis*, Simon Fraser University, Canada, Dec. 1992.

[9] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv.*, 19:201–260, 1987.

[10] T. Imielinski. Intelligent query answering in rule based systems. *J. Logic Programming*, 4:229–257, 1987.

[11] K. A. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 449–462. AAAI/MIT Press, 1991.

[12] J. King. QUIST: A system for semantic query optimization in relational databases. In *Proc. 7th Int. Conf. Very Large Data Bases*, pages 510–517, Cannes, France, 1981.

[13] L. V. S. Lakshmanan and R. Missaoui. On semantic query optimization in deductive databases. In *Proc. 8th Int. Conf. Data Engineering*, pages 368–375, Phoenix, AZ, Feb. 1992.

[14] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et. al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.

[15] A. Motro. Using integrity constraints to provide intensional responses to relational queries. In *Proc. 15th Int. Conf. Very Large Data Bases*, pages 237–246, Amsterdam, Netherlands, Aug. 1989.

[16] A. Motro and Q. Yuan. Querying database knowledge. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 173–183, Atlantic City, NJ, June 1990.

[17] D. E. O'Leary. Knowledge discovery as a threat to database security. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 507–516. AAAI/MIT Press, 1991.

[18] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

[19] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

[20] A. Pirotte and D. Roelants. Constraints for improving the generation of intensional answers in a deductive database. In *Proc. 5th Int. Conf. Data Engineering*, pages 652–659, Los Angeles, CA, Feb. 1989.

[21] C-D. Shum and R. Muntz. Implicit representation for extensional answers. In *Proc. 2nd Int. Conf. Expert Database Systems*, pages 497–522, Vienna, VA, April 1988.

[22] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. ACM*, 34:94–109, 1991.

[23] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18:197–222, 1986.

[24] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vols. 1 & 2.* Computer Science Press, 1989.