Knowledge Discovery in Databases: An Attribute-Oriented Approach

Jiawei Han†, **Yandong Cai**, and **Nick Cercone**†
School of Computing Science
Simon Fraser University
Burnaby, British Columbia, Canada V5A 1S6
{ han, cai, nick }@cs.sfu.ca

Abstract

Knowledge discovery in databases, or data mining, is an important issue in the development of data- and knowledge-base systems. 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 set-oriented database operations and extracts generalized data from actual data in databases. An attribute-oriented concept tree ascension technique is applied in generalization, which substantially reduces the computational complexity of database learning processes. Different kinds of knowledge rules, including characteristic rules, discrimination rules, quantitative rules, and data evolution regularities can be discovered efficiently using the attribute-oriented approach. In addition to learning in relational databases, the approach can be applied to knowledge discovery in nested relational and deductive databases. Learning can also be performed with databases containing noisy data and exceptional cases using database statistics. Furthermore, the rules discovered can be used to query database knowledge, answer cooperative queries and facilitate semantic query optimization. Based upon these principles, a prototyped database learning system, DBLEARN, has been constructed for experimentation.

1. Introduction

Knowledge discovery is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [7]. 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. Recently, data mining has been ranked as one of the most promising research topics for the 1990s by both database and machine learning researchers [7, 20].

In our previous studies [1, 10], an attribute-oriented induction method has been developed for knowledge discovery in relational databases. The method integrates a machine learning paradigm, especially *learning-from-examples* techniques, with database operations and extracts generalized data from actual data in databases. A key to our approach is the attribute-oriented concept tree ascension for generalization which applies well-developed set-oriented database operations and substantially reduces the computational complexity of the database learning processes.

In this paper, the attribute-oriented approach is developed further for learning different kinds of knowledge rules, including characteristic rules, discrimination rules, quantitative rules, and data evolution regularities. Moreover, in addition to learning from relational databases, this approach is extended to knowledge discovery in other kinds of databases, such as nested relational and deductive databases. Learning can also be performed with databases containing noisy data and exceptional cases using database statistics. Furthermore, the rules discovered can be used for querying database knowledge, cooperative query answering and semantic query optimization.

The paper is organized as follows. Primitives for knowledge discovery in databases are introduced in Section 2. The principles of attribute-oriented induction are presented in Section 3. The discovery of different kinds of knowledge rules in relational systems is considered in

Proceedings of the 18th VLDB Conference

Vancouver, British Columbia, Canada 1992

† The work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Research grants OPG0037230/OPG0043090 and a research grant from Centre for Systems Science of Simon Fraser University.

Section 4. Extension of the method to extended-relational databases is discussed in Section 5. A comparison of our approach with other learning algorithms is contained in Section 6. The application of discovered rules to enhance system performance is discussed in Section 7, and we summarize our investigation in Section 8.

2. Primitives for Knowledge Discovery in Databases

Three primitives should be provided for the specification of a learning task: *task-relevant data, background knowledge*, and *the expected representations of learning results*. For illustrative purposes, we first examine relational databases, however, the results are generalized to other kinds of databases in later discussions.

2.1. Data relevant to the discovery process

A database usually stores a large amount of data, of which only a portion may be relevant to a specific learning task. For example, to characterize the features of *graduate students* in *science*, only the data relevant to *graduates* in *science* are appropriate in the learning process. Relevant data may extend over several relations. A query can be used to collect task-relevant data from the database.

Task-relevant data can be viewed as examples for learning processes. Undoubtedly, *learning-from-examples* [9, 14] should be an important strategy for knowledge discovery in databases. Most *learning-from-examples* algorithms partition the set of examples into *positive* and *negative* sets and perform *generalization* using the positive data and *specialization* using the negative ones [14]. Unfortunately, a relational database does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies mainly on generalization, which should be performed cautiously to avoid over-generalization.

Many kinds of rules, such as *characteristic rules*, *discrimination rules*, *data evolution regularities*, etc. can be discovered by induction processes. A **characteristic rule** is an assertion which characterizes a concept satisfied by all or a majority number of the examples in the class undergoing learning (called the **target class**). For example, the symptoms of a specific disease can be summarized by a characteristic rule. A **discrimination rule** is an assertion which discriminates a concept of the class being learned (the **target class**) from other classes (called **contrasting classes**). For example, to distinguish one disease from others, a discrimination rule should summarize the symptoms that discriminate this disease from others. Furthermore, **data evolution regularity** represents the *characteristics* of the changed data if it is a characteristic rule, or the features which *discriminate* the current data instances from the previous ones if it is a discrimination rule. If quantitative measurement is associated with a learned rule, the rule is called a **quantitative rule**.

In learning a characteristic rule, relevant data are collected into one class, the target class, for generalization. In learning a discrimination rule, it is necessary to collect data into two classes, the target class and the contrasting class(es). The data in the contrasting class(es) imply that such data cannot be used to distinguish the target class from the contrasting ones, that is, they are used to exclude the properties shared by both classes.

2.2. Background knowledge

Concept hierarchies represent necessary background knowledge which controls the generalization process. Different levels of concepts are often organized into a taxonomy of concepts. The concept taxonomy can be partially ordered according to a general-to-specific ordering. The most general concept is the null description (described by a reserved word "*ANY*"), and the most specific concepts correspond to the specific values of attributes in the database [11]. Using a concept hierarchy, the rules learned can be represented in terms of generalized concepts and stated in a simple and explicit form, which is desirable to most users.

{ biology, chemistry, computing, ..., physics } $\subset$ science
{ literature , music, ..., painting } $\subset$ art
{ science, art } $\subset$ ANY (major)
{ freshman, sophomore, junior, senior } $\subset$ undergraduate
{ M.S., M.A., Ph.D. } $\subset$ graduate
{ undergraduate, graduate } $\subset$ ANY (status)
{ Burnaby, ..., Vancouver, Victoria } $\subset$ British Columbia
{ Calgary, ..., Edmonton, Lethbridge } $\subset$ Alberta
{ Hamilton, Toronto, ..., Waterloo } $\subset$ Ontario
{ Bombay, ..., New Delhi } $\subset$ India
{ Beijing, Nanjing, ..., Shanghai } $\subset$ China
{ China, India, Germany, ..., Switzerland } $\subset$ foreign
{ Alberta, British Columbia, ..., Ontario } $\subset$ Canada
{ foreign, Canada } $\subset$ ANY (place)
{ 0.0 — 1.99 } $\subset$ poor
{ 2.0 — 2.99 } $\subset$ average
{ 3.0 — 3.49 } $\subset$ good
{ 3.5 — 4.0 } $\subset$ excellent
{ poor, average, good, excellent } $\subset$ ANY (grade)

Figure 1. A concept hierarchy table of the database

Example 1. The *concept hierarchy table* of a typical university database is shown in Fig. 1, where $A \subset B$ indicates that B is a *generalization* of A. A concept tree represents a taxonomy of concepts of the values in an attribute domain. A concept tree for *status* is shown in Fig. 2. □

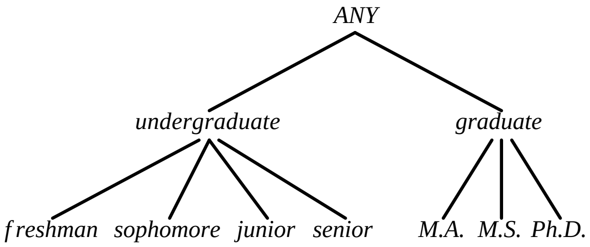


Figure 2. A concept tree for *status*.

Concept hierarchies can be provided by knowledge engineers or domain experts. This is reasonable for even large databases since a concept tree registers only the *distinct* discrete attribute values or ranges of numerical values for an attribute which are, in general, not very large and can be input by domain experts. Moreover, many conceptual hierarchies are actually stored in the database implicitly. For example, the information that "*Vancouver is a city of British Columbia, which, in turn, is a province of Canada*", is usually stored in the database if there are "*city*", "*province*" and "*country*" attributes. Such hierarchical relationships can be made explicit at the schema level by indicating "*city* $\subset$ *province* $\subset$ *country*". Then, the taxonomy of all the cities stored in the database can be retrieved and used in the learning process.

Some concept hierarchies can be *discovered* automatically or semi-automatically. Numerical attributes can be organized as discrete hierarchical concepts, and the hierarchies can be constructed automatically based on database statistics. For example, for an attribute "*GPA*", an examination of the attribute value distribution in the database may disclose that *GPA* falls between 0 to 4, and most *GPA*'s for graduates are clustered between 3 and 4. One may classify 0 to 1.99 into one class, and 2 to 2.99 into another but give finer classifications for those between 3 and 4. Even for attributes with discrete values, statistical techniques can be used under certain circumstances [6]. For example, if the birth-place of most students are clustered in Canada and scattered in many different countries, the highest level concepts of the attribute can be categorized into "*Canada*" and "*foreign*". Similarly, an available concept hierarchy can be modified based on database statistics. Moreover, the concept hierarchy of an attribute can also be discovered or refined based on its relationship with other attributes [6].

Different concept hierarchies can be constructed on the same attribute based on different viewpoints or preferences. For example, the birthplace could be organized according to administative regions such as provinces, countries, etc., geographic regions such as east-coast, west-coast, etc., or the sizes of the city, such as, metropolis, small-city, town, countryside, etc. Usually, a commonly referenced concept hierarchy is associated with an attribute as the default one. Other hierarchies can be chosen explicitly by preferred users in the learning process.

2.3. Representation of learning results

From a logical point of view, each tuple in a relation is a logic formula in conjunctive normal form, and a data relation is characterized by a large set of disjunctions of such conjunctive forms. Thus, both the data for learning and the rules discovered can be represented in either relational form or first-order predicate calculus.

A relation which represents *intermediate* (or *final*) learning results is called an *intermediate* (or a *final*) **generalized relation**. In a generalized relation, some or all of its attribute values are generalized data, that is, nonleaf nodes in the concept hierarchies. Some learning-from-examples algorithms require the final learned rule to be in conjunctive normal form [14]. This requirement is usually unreasonable for large databases since the generalized data often contain different cases. However, a rule containing a large number of disjuncts indicates that it is in a complex form and further generalization should be performed. Therefore, the final generalized relation should be represented by either one tuple (a conjunctive rule) or a small number (usually 2 to 8) of tuples corresponding to a disjunctive rule with a small number of disjuncts. A system may allow a user to specify the preferred **generalization threshold**, a maximum number of disjuncts of the resulting formula. For example, if the threshold value is set to three, the final generalized rule will consist of *at most* three disjuncts.

The complexity of the rule can be controlled by the generalization threshold. A moderately large threshold may lead to a relatively complex rule with many disjuncts and the results may not be fully generalized. A small threshold value leads to a simple rule with few disjuncts. However, small threshold values may result in an overly generalized rule and some valuable information may get lost. A better method is to adjust the threshold values within a reasonable range *interactively* and to select the best generalized rules by domain experts and/or users.

Exceptional data often occur in a large relation. It is important to consider exceptional cases when learning in databases. Statistical information helps *learning-from-examples* to handle exceptions and/or noisy data [3, 15]. A special attribute, *vote*, can be added to each generalized relation to register the number of tuples in the original relation which are generalized to the current tuple in the generalized relation. The attribute *vote* carries database statistics and supports the pruning of scattered data and the generalization of the concepts which take a majority of votes. The final generalized rule will be the rule which represents the characteristics of a *majority* number of facts in the database (called an *approximate rule*) or indicates *quantitative* measurement of each conjunct or disjunct in the rule (called a *quantitative rule*).

2.4. A database learning language

Given a number of examples, generalization can be performed in many different directions [5]. Unconstrained learning may result in a very large set of learned rules. Moreover, different rules can be extracted from the same set of data using different background knowledge (concept hierarchies). In order to constrain a generalization process and extract interesting rules from databases, learning should be directed by specific learning requests and background knowledge.

A database learning request should consist of (i) a database query which extracts the relevant set of data, (ii) the kind of rules to be learned, (iii) the specification of the target class, and possibly, the contrasting classes depending on the rules to be learned, (iv) the preferred concept hierarchies, and (v) the preferred form to express learning results. Notice that (iv) and (v) are optional since default concept hierarchies and a default generalization threshold can be used if no preference is specified explicitly.

A database learning system, **DBLEARN**, has been proposed in our study [1]. The language of **DBLEARN** can be viewed as an extension to the relational language SQL for knowledge discovery in databases. Because of limited space, we present one short illustrative example of a learning request specified to **DBLEARN**.

Example 2. Our objective is to learn a discrimination rule which distinguishes Ph.D. students from M.S. students in science based upon the level of courses in science in which they assist. The learning involves both *Student* and *Course* relations. The request is specified to **DBLEARN** as follows.

in relation *Student S, Course C*
learn discrimination rule for *S.Status = "Ph.D."*
in contrast to *S.Status = "M.S."*
where *S.Major = "science"* **and** *C.Dept = "science"*
    and *C.TA = S.Name*
in relevance to *C.Level*

Notice that a database query is embedded in the learning request, and "*science*" is a piece of generalized data which can be found in the concept hierarchy table. The preferred conceptual hierarchy can be specified by "**using hierarchy** *hierarchy_name*"; and the preferred generalization threshold can be specified by "**using threshold:** *threshold_value*". Since neither of them is specified explictly in this learning request, default hierarchies and thresholds are used. □

3. Principles for Attribute-Oriented Induction in Relational Databases

3.1. Basic strategies for attribute-oriented induction

A set of seven basic strategies are defined for performing attribute-oriented induction in relational databases, which are illustrated using Example 3.

Example 3. Let Table 1 be the relation *student* in a *university* database.

Name	Status	Major	Birth_Place	GPA
Anderson	M.A.	history	Vancouver	3.5
Bach	junior	math	Calgary	3.7
Carlton	junior	liberal arts	Edmonton	2.6
Fraser	M.S.	physics	Ottawa	3.9
Gupta	Ph.D.	math	Bombay	3.3
Hart	sophomore	chemistry	Richmond	2.7
Jackson	senior	computing	Victoria	3.5
Liu	Ph.D.	biology	Shanghai	3.4
...	...	...	...	...
Meyer	sophomore	music	Burnaby	3.0
Monk	Ph.D.	computing	Victoria	3.8
Wang	M.S.	statistics	Nanjing	3.2
Wise	freshman	literature	Toronto	3.9

Table 1. A relation *Student* in a *university* database.

Suppose that the learning task is to learn characteristic rules for graduate students relevant to the attributes *Name, Major, Birth_Place* and *GPA*, using the default conceptual hierarchy presented in Fig. 1 and the default threshold value of 3. The learning task is represented in **DBLEARN** as follows.

in relation *Student*
learn characteristic rule for *Status = "graduate"*
in relevance to *Name, Major, Birth_Place, GPA*

Name	Major	Birth_Place	GPA	*vote*
Anderson	history	Vancouver	3.5	1
Fraser	physics	Ottawa	3.9	1
Gupta	math	Bombay	3.3	1
Liu	biology	Shanghai	3.4	1
...	...	...	...	...
Monk	computing	Victoria	3.8	1
Wang	statistics	Nanjing	3.2	1

Table 2. The initial data relation for induction.

For this learning request, preprocessing is performed by first selecting *graduate* students. Since "*graduate*" is a nonleaf node in the concept hierarchy on Status, the hierarchy table should be consulted to extract the set of the corresponding primitive data stored in the relation, which is {*M.S.*, *M.A.*, *Ph.D.*}. Then the data about graduates can be retrieved and projected on relevant attributes *Name*, *Major*, *Birth_Place*, and *GPA*, which results in an initial data relation on which induction can be performed. Table 2 reflects the result of this preprocessing and a special attribute *vote* is attached to each tuple with its initial value set to 1. Such a preprocessed data relation is called an **initial relation**.

Attribute-oriented induction is performed on the initial relation.

Strategy 1. (**Generalization on the smallest decomposable components**) *Generalization should be performed on the smallest decomposable components (or attributes) of a data relation*.

Rationale. Generalization is a process of learning from positive examples. Generalization on the smallest decomposable components rather than on composite attributes ensures that the smallest possible chance is considered in the generalization, which enforces the *least commitment principle* (commitment to minimally generalized concepts) and avoids over-generalization. □

We examine the task-relevant attributes in sequence. There is no higher level concept specified on the first attribute *Name*. Thus, the attribute should be removed in generalization, which implies that general properties of a *graduate student* cannot be characterized by the attribute *Name*. This notion is based on Strategy 2.

Strategy 2. (**Attribute removal**) *If there is a large set of distinct values for an attribute but there is no higher level concept provided for the attribute, the attribute should be removed in the generalization process*.

Rationale. This strategy corresponds to the generalization rule, *dropping conditions*, in *learning-from-examples* [14]. Since an attribute-value pair represents a conjunct in the logical form of a tuple, removal of a conjunct eliminates a constraint and thus generalizes the rule. If there is a large set of distinct values in an attribute but there is no higher level concept provided for it, the values cannot be generalized using higher level concepts, thus the attribute should be removed. □

The three remaining attributes *Major*, *Birth_place* and *GPA* can be generalized by substituting for subordinate concepts by their corresponding superordinate concepts. For example, "*physics*" can be substituted for by "*science*" and "*Vancouver*" by "*B.C.*". Such substitutions are performed attribute by attribute, based on Strategy 3.

Strategy 3. (**Concept tree ascension**) *If there exists a higher level concept in the concept tree for an attribute value of a tuple, the substitution of the value by its higher level concept generalizes the tuple. Minimal generalization should be enforced by ascending the tree one level at a time*.

Rationale. This strategy corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [14]. 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. Ascending the concept tree one level at a time ensures that the generalization shall follow the least commitment principle and thus reduces chances of over-generalization. □

As a result of concept tree ascension, different tuples may generalize to an identical tuple where two tuples are **identical** if they have the same corresponding attribute values without considering the special attribute *vote*. To incorporate quantitative information in the learning process, *vote* should be accumulated in the generalized relation when merging identical tuples.

Strategy 4. (**Vote propagation**) *The value of the vote of a tuple should be carried to its generalized tuple and the votes should be accumulated when merging identical tuples in generalization.*

Rationale. Based on the definition of *vote*, the vote of each generalized tuple must register the number of the tuples in the initial data relation generalized to the current one. Therefore, to keep the correct votes registered, the vote of each tuple should be carried in the generalization process, and such votes should be accumulated when merging identical tuples. □

By removing one attribute and generalizing the three remaining ones, the relation depicted in Table 2 is generalized to a new relation as illustrated in Table 3.

Major	Birth_Place	GPA	*vote*
art	B.C.	excellent	35
science	Ontario	excellent	10
science	B.C.	excellent	30
science	India	good	10
science	China	good	15

Table 3. A generalized relation

To judge whether an attribute needs to be further generalized, we have

Strategy 5. (**Threshold control on each attribute**) *If the number of distinct values of an attribute in the target class is larger than the generalization threshold value, further generalization on this attribute should be performed.*

Rationale. The generalization threshold controls and represents the maximum number of tuples of the target class in the final generalized relation. If one attribute contains more distinct values than the threshold, the number of distinct tuples in the generalized relation must be greater than the threshold value. Thus the values in the attribute should be further generalized. □

After attribute-oriented ascension of concept trees and the merging of identical tuples, the total number of tuples in a generalized relation may remain greater than the specified threshold. In this case, further generalization is required. Strategy 6 has been devised for this generalization.

Strategy 6. (**Threshold control on generalized relations**) *If the number of tuples of a generalized relation in the target class is larger than the generalization threshold value, further generalization on the relation should be performed.*

Rationale. Based upon the definition of the generalization threshold, further generalization should be performed if the number of tuples in a generliazed relation is larger than the threshold value. By further generalization on selected attribute(s) and merging of identical tuples, the size of the generalized relation will be reduced. Generalization should continue until the number of remaining tuples is no greater than the threshold value. □

At this stage, there are usually alternative choices for selecting a candidate attribute for further generalization. Criteria, such as the preference of a larger reduction ratio on the number of tuples or the number of distinct attribute values, the simplicity of the final learned rules, etc., can be used for selection. Interesting rules can often be discovered by following different paths leading to several generalized relations for examination, comparison and selection. Following different paths corresponds to the way in which different people may learn differently from the same set of examples. The generalized relations can be examined by users or experts *interactively* to filter out trivial rules and preserve interesting ones [23].

Table 3 represents a generalized relation consisting of five tuples. Further generalization is needed to reduce the number of tuples. Since the attribute "*Birth_Place*" contains four distinct values, generalization should be performed on it by ascending one level in the concept tree, which results in the relation shown in Table 4.

Major	Birth_Place	GPA	*vote*
art	Canada	excellent	35
science	Canada	excellent	40
science	foreign	good	25

Table 4. Further generalization of the relation.

The *final generalized relation* consists of only a small number of tuples, and this generalized relation can be transformed into a simple logical formula. Based upon the principles of logic and databases [8, 22], we have evolved Strategy 7.

Strategy 7. (**Rule transformation**) *A tuple in a final generalized relation is transformed to conjunctive normal form, and multiple tuples are transformed to disjunctive normal form*.

Notice that simplification can be performed on Table 4 by *unioning* the first two tuples if set representation of an attribute is allowed. The relation so obtained is shown in Table 5.

Major	Birth_Place	GPA	*vote*
{ art, science }	Canada	excellent	75
science	foreign	good	25

Table 5. Simplification of the generalized relation.

Suppose *art* and *science* cover all of the *Major* areas. Then {*art*, *science*} can be generalized to *ANY* and be removed from the representation. Therefore, the final generalized relation is equivalent to rule (*1*), that is, *a graduate is either (with 75% probability) a Canadian with an excellent GPA or (with 25% probability) a foreign student, majoring in sciences with a good GPA.* Notice that since a characteristic rule characterizes all of the data in the target class, its then-part represents the necessary condition of the class.

(*1*) $\forall(x)\ graduate(x) \rightarrow (\ Birth\_Place(x) \in\ Canada\ \wedge$
$GPA(x) \in\ excellent\ )\ [75\%]\ \vee\ (\ Major(x) \in\ science\ \wedge$
$Birth\_Place(x) \in\ foreign\ \wedge\ GPA(x) \in\ good\ )\ [25\%].$

Rule (*1*) is a *quantitative rule*. It can also be expressed in the *qualitative form* by dropping the quantitative measurement. Moreover, the learning result may also be expressed as an *approximate rule* by dropping the conditions or conclusions with negligible probabilities.

3.2. Basic attribute-oriented induction algorithm

The basic idea of attribute-oriented induction is summarized in the following algorithm.

Algorithm 1. *Basic attribute-oriented induction in relational databases.*

Input: (i) A relational database, (ii) the learning task, (iii) the (optional) preferred concept hierarchies, and (iv) the (optional) preferred form to express learning results (e.g., generalization *threshold*).

Output. A characteristic rule learned from the database.

Method. Basic attribute-oriented induction consists of the following four steps:

Step 1. *Collect the task-relevant data*,

Step 2. *Perform basic attribute-oriented induction*,

Step 3. *Simplify the generalized relation,* and

Step 4. *Transform the final relation into a logical rule.*

Notice that *Step 2* is performed as follows.

begin {*basic attribute-oriented induction*}

for each attribute A_i ($1 \le i \le n$, where n = # of attributes) in the generalized relation GR **do**

while *#_of_distinct_values_in_A_i* > *threshold* **do** {

if no higher level concept in the concept hierarchy table for A_i

then remove A_i

else substitute for the values of the A_i's by its corresponding minimal generalized concept;

merge identical tuples }

while *#_of_tuples in GR* > threshold **do** {

selectively generalize attributes;

merge identical tuples }

end. □

Theorem 1. Algorithm 1 learns correctly characteristic rules from relational databases.

Proof.

In Step 1, the relevant set of data in the database is collected for induction. The then-part in the first while-loop of Step 2 incorporates Strategy 1 (attribute removal), and the else-part utilizes Strategy 3 (concept tree ascension). The condition for the first while-loop is based on Strategy 5 and that for the second one on Strategy 6 (threshold control). Strategy 2 is used in Step 2 to ensure that generalization is performed on the minimal decomposable components. Each generalization statement in both while-loops applies the least-commitment principle based on those strategies. Finally, Steps 3 and 4 apply logic transformations based on the correspondence between relational tuples and logical formulas. Thus the obtained rule should be the desired result which summarizes the characteristics of the target class. □

The basic attribute-oriented induction algorithm extracts a characteristic rule from an initial relation. Since the generalized rule covers all of the positive examples in the database, it forms the necessary condition of the learning concept, that is, the rule is in the form of

$$learning\_class(x) \rightarrow condition(x),$$

where $condition(x)$ is a formula containing x. However, since data in other classes are not taken into consideration in the learning process, there could be data in other classes which also meet the specified condition. Thus, $condition(x)$ is necessary but may not be sufficient for x to be in the learning class.

4. Learning Other Knowledge Rules by Attribute-Oriented Induction

The attribute-oriented induction method can also be applied to learning other knowledge rules, such as discrimination rules, data evolution regularities, etc.

4.1. Learning discrimination rules

Since a discrimination rule distinguishes the concepts of the target class from those of contrasting classes, the generalized condition in the target class that overlaps the condition in contrasting classes should be detected and removed from the description of discrimination rules. Thus, a discrimination rule can be extracted by generalizing the data in both the target class and the contrasting class synchronously and excluding properties that overlap in both classes in the final generalized rule.

To implement this notion, the basic attribute-oriented algorithm can be modified correspondingly for discovery of discrimination rules. We illustrate the process with Example 4.

Example 4. Suppose a discrimination rule is to be extracted to distinguish graduate students from undergraduates in the relation *Student* (Table 1). Clearly, both the target class *graduate* and the contrasting class *undergraduate* are relevant to the learning process, and the data should be partitioned into two portions: *graduate* in contrast to *undergraduate*. Generalization can be performed synchronously in both classes by *attribute removal* and *concept tree ascension*. Suppose the relation is generalized to Table 6.

Class	Major	Birth_Place	GPA	*vote*	*mark*
graduate	art	B.C.	excellent	35	
	science	Ontario	excellent	10	
	science	B.C.	excellent	30	*
	science	India	good	10	
	science	China	good	15	
undergrad.	science	Alberta	excellent	15	
	art	Alberta	average	20	
	science	B.C.	average	60	
	science	B.C.	excellent	35	*
	art	B.C.	average	50	
	art	Ontario	excellent	20	

Table 6. A generalized relation

As shown in Table 6, different classes may share tuples. The tuples shared by different classes are called *overlapping tuples*. Obviously, the third tuple of the class *graduate* and the fourth tuple of the class *undergrad.* are overlapping tuples, which indicates that a B.C. born student, majoring in science with excellent GPA, may or may not be a graduate student. In order to get an effective discrimination rule, care must be taken to handle the overlapping tuples. We utilize Strategy 8.

Strategy 8. (**Handling overlapping tuples**) *If there are* *overlapping tuples in both the target and contrasting classes, these tuples should be marked and be excluded from the final discrimination rule.*

Rationale. Since the overlapping tuples represent the same assertions in both the target class and the constrasting class, the concept described by the overlapping tuples cannot be used to distinguish the target class from the contrasting class. By detecting and marking overlapping tuples, we have the choice of including only those assertions which have a discriminating property in the rule, which ensures the correctness of the learned discrimination rule. □

After marking the third tuple in the class of *graduate* and the fourth tuple in the class of *undergrad.*, the target class contains four unmarked tuples as shown in Table 6, which implies that the resulting rule will contain four disjuncts. Suppose the threshold value is 3, further generalization is performed on attribute "*Birth_Place*", which results in the relation shown in Table 7.

Class	Major	Birth_Place	GPA	*vote*	*mark*
graduate	art	Canada	excellent	35	*
	science	Canada	excellent	40	*
	science	foreign	good	25	
undergrad.	science	Canada	excellent	50	*
	arts	Canada	average	70	
	science	Canada	average	60	
	art	Canada	excellent	20	*

Table 7. A generalized relation

Notice that overlapping marks should be inherited in their generalized tuples because their generalized concepts still overlap with that in the contrasting class. Moreover, since generalization may produce new overlapping tuples, an overlapping check should be performed at each ascension of the concept tree. The generalization process repeats until the number of unmarked tuples in the target class is below the specified threshold value.

Since Table 7 has one unmarked tuple and two marked tuples in the target class, the *qualitative* discrimination rule should contain only the unmarked tuple, as shown in rule (*2a*).

(*2a*) $\forall$ *(x) graduate(x)* ←
Major(x) ∈ *science* /\ *Birth_Place(x)* ∈ *foreign* /\ *GPA(x)* ∈ *good.*

Rule (*2a*) is a qualitative rule which excludes overlapping disjuncts. In many cases, however, it is informative to derive a quantitative rule from the final generalized relation, which associates with each disjunct a quantitative measurement (called *d-weight*) to indicate its discriminating ability.

Definition. Let q be a generalized concept (tuple) and C_j be the target class. The **d-weight** for q (referring to the target class) is the ratio of the number of original tuples in the target class covered by q to the total number of tuples in both the target class and the contrasting classes covered by q. Formally, the d-weight of the concept q in class C_j is defined as below.

$$d\_weight \ = \ votes\,(q \in C_j) \,/ \sum_{i=1}^{K} votes\ (q \in C_i),$$

where K stands for the total number of the target and contrasting classes, and C_j is in $\{C_1, ..., C_K\}$.

The range for d-weight is in the interval [0 – 1]. A high d-weight indicates that the concept is primarily derived from the target class C_j, and a low d-weight implies that the concept is primarily derived from the contrasting class(es).

The d-weight for the first tuple in the target class is 35/(35 + 20) = 63.64%, and that for the second and the third tuples are 44.44% and 100%. Notice that the d-weight for any unmarked tuple is 100%. The quantitative discrimination rule for *graduates* can be expressed as (*2b*).

(*2b*) $\forall$ *(x) graduate(x)* ←
(Major(x) ∈ *science* /\ *Birth_Place(x)* ∈ *foreign* /\ *GPA(x)* ∈ *good) [100%]* \/
(Major(x) ∈ *art* /\ *Birth_Place(x)* ∈ *Canada* /\ *GPA(x)* ∈ *excellent) [63.64%]* \/
(Major(x) ∈ *science* /\ *Birth_Place(x)* ∈ *Canada* /\ *GPA(x)* ∈ *excellent) [44.44%].*

Rule (*2b*) implies that *if a foreign student majors in sciences with a good GPA, (s)he is certainly a graduate; if a Canadian student majors in art with excellent GPA, the probability for him (her) to be a graduate is 63.64%; and if a Canadian student majors in science with excellent GPA, the probability for him (her) to be a graduate is 44.44%*.

A qualitative discrimination rule provides a sufficient condition but not a necessary one for an object (or a tuple) to be in the target class. Although those tuples which meet the condition are in the target class, those in the target class may not necessarily satisfy the condition since the rule may not cover *all* of the positive examples of the target class in the database. Thus, the rule should be presented in the form of

learning_class(x) ← *condition(x).*

A quantitative discrimination rule presents the quantitative measurement of the properties in the target class versus that in the contrasting classes. A 100% d-weight indicates that the generalized tuple is in the target class only. Otherwise, the d-weight shows the possibilities for a generalized tuple to be in the target class.

4.2. Learning data evolution regularities

Data evolution regularity reflects the trend of changes in a database over time. Discovery of regularities in an evolving database is important for many applications. To simplify our discussion, we assume that the database schema remains stable in data evolution. A **database instance**, DB_t, is the database state, i.e., all of the data in the database, at time t. At least two different database instances, DB_{t_1} and DB_{t_2} where $t_1 \neq t_2$, are required for such knowledge discovery. Our discussion can be generalized to multiple (> 2) database instances.

Data evolution regularities can be classified into characteristic rules and discrimination rules. The former rules summarize characteristics of the changed data; while the latter distinguish general charteraristics of the relevant data in the current database from those in a previous database instance. We show that attribute-oriented induction can be used in the generalization process.

Example 5. Let the learning request be to *find the characteristics of those graduate students whose GPA increases at least 0.5 in the last six months*.

The knowledge discovery process can be partitioned into two phases: (1) collecting task-relevant data, and (2) performing attribute-oriented induction on the relevant data. The first phase is performed by finding all of the graduate students whose GPA increases at least 0.5 in the last six months based upon the current database instance and the instance six months ago. Since *graduate* is a nonprimitive concept, data retrieval should be performed by consulting the concept hierarchy table as well. The second phase is carried out in the same manner as the previously studied attribute-oriented induction for learning characteristic rules.

Suppose that another learning request is to *distinguish the characteristics of the undergraduate students enrolled in July 1990 from those enrolled in July 1992*.

The knowledge discovery process can still be partitioned into two phases: (1) collecting the task-relevant data and grouping them into two classes, the target class and the contrasting class, and (2) performing attribute-oriented induction synchronously on the two classes. The first phase is performed by finding all of the undergraduate students enrolled in July 1990 and those enrolled in July 1992 and grouping them into two classes respectively. The second phase is the same as the previously studied attribute-oriented induction for learning discrimination rules. □

Such a process can also be used to study the general characteristics of the newly inserted or newly deleted data sets in a database. In general, data evolution regularities can be extracted by collecting the learning task-relevant data (usually, the evolving portion) in different database instances and performing attribute-oriented induction on the corresponding task-relevant data sets.

5. Towards Knowledge Discovery in Extended-Relational Databases

The relational data model has been extended in many ways to meet the requirements of new database applications [20]. Nested-relational and deductive databases are two influential extended-relational systems. Interestingly, attribute-oriented induction can be easily extended to knowledge discovery in these systems.

The nested relational model allows nonatomic, relational-valued domains. Thus, a hierarchy of nested relations is formed [18]. The attribute-oriented induction can be performed easily on the atomic domains in the same way as in relational systems. The induction can be performed on the non-atomic domains in two different ways: (1) *unnesting* the nested relations, which transforms nested relations into *flat* relations to which the previously described method can be applied; or (2) *performing induction directly* on the nested relations by treating the nonatomic domains as set-valued domains.

Example 6. Let the student relation in the *university* database contain one more attribute *hobby*, which registers a set of hobbies for each student. We study the learning request: *discover the relationship between GPA and hobby for graduate students in computing science*.

Although generalization can be performed by first flattening the (*nested*) relation, *Student*, into a nonnested relation and then performing attribut-oriented induction, it is more efficient and elegant to perform induction directly on the attribute *hobby* by treating the domain as a set-valued domain. For example, without flattening the relation, the set {*badminton, violin*} can be generalized to a new set {*sports, music*}. A benefit of direct induction on set-valued attribute is that, unlike *unnesting*, it does not flatten an original tuple into *several* tuples. Vote accumulation can be handled directly when merging identical tuples in the concept tree ascension since one generalized tuple corresponds to one original tuple in the generalized relation. □

Deductive database systems can be viewed as another extension to relational systems by introducing deduction power to relational databases. By incorporating deduction rules and integrity constraints in the generalization process, attribute-oriented induction can be performed on deductive databases as well. There are two cases to be considered for knowledge extraction in deductive databases: (1) discovery of knowledge rules on the relations defined by deduction rules, and (2) discovery of new knowledge rules from the existing rules.

In the first case, deduction can be performed by executing a deductive query which retrieves task-relevant data from databases. Then induction can be performed on the deduced data set.

Example 7. Let a deductive database be constructed from the relational database *university*, with *award candidate* defined by a set of deduction rules $\{(3a), (3b)\}$.

(3a) $award\_candidate(X) \leftarrow$
$status(X) = graduate,\ gpa(X) \geq 3.75.$

(3b) $award\_candidate(X) \leftarrow$
$status(X) = undergraduate,\ gpa(X) \geq 3.5.$

Suppose that the learning request is to *find general characteristics of award candidates in the computing science department*. In a manner similar to the previously described process, the knowledge discovery process can be partitioned into two phases: (1) collecting the task-relevant data, and (2) performing attribute-oriented induction. The first phase is performed by finding all of the award candidates in the computing science department. This corresponds to a dedutive query which can be processed using deductive database technology [8, 22]. The second phase is to extract knowledge from those candidates, which is the same as attribute-oriented induction for learning characteristic rules. □

When deduction rules involve recursion, the deduction process itself could be rather complex [22]. However, the knowledge discovery process is still carried out by performing first deduction then induction. Notice that deduction may involve consulting concept hierarchies if a deduction rule is defined using higher-level concepts.

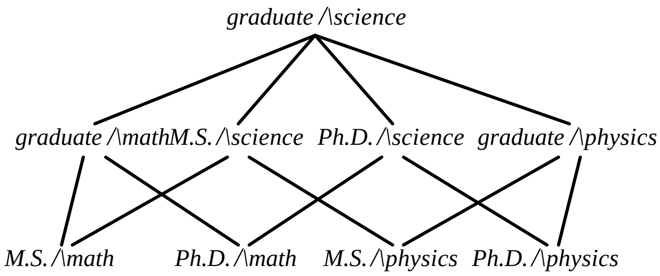
As to the second case, discovering new rules from the existing ones, an existing rule could be an induced rule or a partially induced one. If such a rule represents a generalization of all of the relevant extensional data, further induction can be performed directly on the rule itself. If such a rule, together with some extensional data, determine all of the relevant information in the deductive database, knowledge discovery should be performed on such *hybrid data* by first performing induction on the extensional data to generalize the data to the same level of the concepts as that in the existing rule, and the provided rules are treated as part of the intermediately generalized relation and are merged with the generalized relation. The merged relation can then be further generalized by attribute-oriented induction.

6. A Comparison with Other Learning Algorithms

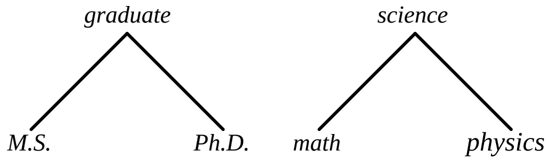
Attribute-oriented induction provides a simple and efficient way to learn different kinds of knowledge rules in relational and extended relational databases. As a newly emerging field, there have been only a few database-oriented knowledge discovery systems reported [7, 12], most of which are based on the previously developed learning algorithms. The major difference of our approach from others is attribute-oriented vs. tuple-oriented induction. It is essential to compare these two approaches.

Both tuple-oriented and attribute-oriented induction take attribute removal and concept tree ascension as their major generalization techniques. However, the former technique performs generalization tuple by tuple, while the latter, attribute by attribute. The two approaches involves significantly different search spaces. Among many *learning-from-examples* algorithms, we use the *candidate elimination* algorithm [16] as an example to demonstrate such a difference.

In the candidate elimination algorithm, the set of all of the concepts which are consistent with training examples is called the *version space* of the training examples. The learning process is the search in the version space to induce a generalized concept which is satisfied by all of the positive examples and none of the negative examples.



(a) The entire version space.



(b) The factored version spaces.

Figure 3. Entire vs. factored version spaces.

Since generalization in an attribute-oriented approach is performed on individual attributes, the concept hierarchy of each attribute can be treated as a factored version space. Factoring the version space may significantly improve the computational efficiency. Suppose there are p nodes in each concept tree and there are k concept trees (attributes) in the relation, the total size of k factorized version spaces is $p \times k$. However, the size of the unfactorized version space for the same concept tree should be p^k [21]. This can be verified from Fig. 3. Suppose the concept hierarchy is specified as: $\{$*math*, *physics*$\} \subset$ *science*, and $\{$*M.S.*, *Ph.D.*$\} \subset$ *graduate*. The corresponding entire version space and factored version space are Fig. 3 (a) and Fig. 3 (b), respectively.

The entire version space contains $3^2 = 9$ nodes, but the factored version spaces contain only $3 \times 2 = 6$ nodes.

Similar arguments hold for other tuple-oriented learning algorithms. Although different algorithms may adopt different search strategies, the tuple-oriented approach examines the training examples one at a time to induce generalized concepts. In order to discover the most specific concept that is satisfied by all of the training examples, the algorithm must search every node in the search space which represents the possible concepts derived from the generalization on this training example. Since different attributes of a tuple may be generalized to different levels, the number of nodes to be searched for a training example may involve a huge number of possible combinations.

On the other hand, an attribute-oriented algorithm performs generalization on each attribute uniformly for all the tuples in the data relation at the *early* generalization stages. It essentially considers only the factored version space. An algorithm which explores different possible combinations for a large number of tuples during such a generalization process will not be productive since such combinations will be merged in further generalizations. Different possible combinations should be explored only when the relation has been generalized to a relatively small intermediate generalized relation, which corresponds to Strategy 6 in the basic attribute-oriented induction. Notice that only one rule formation technique is provided in our basic algorithm. However, the relatively small intermediate generalized relation obtained by attribute-oriented induction can be treated as a springboard on which different knowledge extarction techniques can be explored to form new rules.

Another obvious advantage of our approach over many other learning algorithms is our integration of the learning process with database operations. Relational systems store a large amount of information in a structured and organized manner and are implemented by well-developed storage and accessing techniques [20]. In contrast to most existing learning algorithms which do not take full advantages of these database facilities [5, 12, 15], our approach primarily adopts relational operations, such as selection, join, projection (extracting relevant data and removing attributes), tuple substitution (ascending concept trees), and intersection (discovering common tuples among classes). Since relational operations are set-oriented and have been implemented efficiently in many existing systems, our approach is not only efficient but easily exported to many relational systems.

Our approach has absorbed many advanced features of recently developed learning algorithms [3, 13]. As shown in our study, attribute-oriented induction can learn disjuctive rules and handle exceptional cases elegantly by incorporating statistical techniques in the learning process. Moreover, when a new tuple is inserted into a database relation, rather than restarting the learning process from the beginning, it is preferable to amend and fortify what was learned from the previous data. Our algorithms can be easily extended to faciliate such *incremental learning* [15]. Let the generalized relation be stored in the database. When a new tuple is inserted into a database, the concepts of the new tuple are first generalized to the level of the concepts in the generalized relation. Then the generalized tuple can be naturally merged into the generalized relation.

In our previous discussion, we assume that every concept hierarchy is organized as a balanced tree, and the primitive concepts in the database reside at the leaves of the tree. Hence generalization can be performed synchronously on each attribute, which generalizes the attribute values at the same lower level to the ones at the same higher level. By minor modification to the basic algorithm, induction can also be performed successfully on unbalanced concept trees and on data residing at different levels of concept trees. In such cases, rather than simply performing generalization on every branch of the tree, we check whether a generalized concept may cover other concepts of the same attribute. If the generalized concept covers a concept several levels down the concept tree, the covered concept is then replaced by the generalized concept, that is, ascending the tree several levels at once. By doing so, concepts at different levels can be handled correctly and efficiently.

As another variation, our method can also handle the concepts organized as lattices. If concepts are organized as a lattice, some single concept may be generalized to more than one concept. These generalized concepts are put into intermediate generalized relations upon which further generalizations are performed as discussed. As a consequence, the size of intermediate generalized relations may increase at some stage in the generalization process because of the effect of a lattice. However, since the generalization is controlled by a generalization threshold, the intermediate generalized relation will eventually shrink in subsequent generalizations.

Furthermore, data sampling and parallelism can be explored in knowledge discovery. Attribute-oriented induction can be performed by *sampling* a subset of data from a huge set of relevant data or by first performing induction *in parallel* on several partitions of the relevant data set and then merging the generalized results.

7. Application of Discovered Rules

Knowledge discovery in databases initiates a new frontier for querying database knowledge, cooperative query answering and semantic query optimization. Lots can be explored using meta-data (such as concept hierarchies) and discovered knowledge. We present a few ideas for this below.

Database knowledge represents the semantic information associated with databases, which includes deduction rules, integrity constraints, concept hierarchies about data and general data characteristics [17]. Cooperative query answering consists of analyzing the intent of query and providing generalized, neighborhood or associated information relevant to the query [4]. Semantic query optimization applies database semantics, integrity constraints and knowledge rules to optimize queries for efficient processing [2].

Previous studies on querying database knowledge and intelligent query answering [17, 19] were focused on rules and integrity constraints in relational or deductive databases. With the availability of knowledge discovery tools, it is straightforward to query general data characteristics and utilize induced rules and concept hierarchies. Such queries can be answered by retrieving discovered rules (if such pieces of information are available) or triggering a discovery process. Moreover, some queries can be rewritten based on the analysis of the concept hierarchies and/or answered cooperatively using generalized rules.

Example 8. To *describe the characteristics of the graduate students in cs (computing science) who were born in Canada with excellent academic performance*, the query can be formulated using a syntax similar to the knowledge queries in [17] as follows.

describe *Student*
where *Status = "graduate"* **and** *Major = "cs"* **and**
Birth_Place = "Canada" **and** *GPA = "excellent"*

Notice that "*graduate*", "*Canada*" and "*excellent*" are not stored as primitive data in the *Student* relation. However, using the concept hierarchy table, the query can be reformulated by substituting for "*graduate*" by "{*M.S., M.A. Ph.D*}", etc. Then the rewritten query can be answered by directly retrieving the discovered rule, if it is stored, or performing induction on the relevant data set. □

It is often useful to store some intermediate generalized relations (based upon the frequency of the encountered queries) to facilitate querying database knowledge. When a knowledge query is submitted to the system, selection and further generalization can be performed on such an intermediate generalized relation instead of on primitive data in the database.

Moreover, semantic query optimization can be performed on queries using database semantics, concept hierarchies and the discovered knowledge rules.

Example 9. Let the knowledge rules discovered in our previous examples be stored in the database, and consider the query: *find all the foreign students (born outside of Canada) majoring in science with GPA between 3.2 to 3.4*.

select *Name*
from *Student*
where *Major = "science"* **and** *Birth_Place != "canada"*
and *GPA >= 3.2* **and** *GPA <= 3.4*

According to rule ($2a$), all of the foreign students majoring in science with a good GPA must be graduate students. Since the condition of rule ($2a$) covers what is inquired, the search should be performed on *graduate students* only, that is, the condition, *Status = "graduate"*, can be appended to the query. Therefore, the query can be optimized if the data is grouped or partitioned according to the *status* of the students. □

8. Conclusions

We investigated an attribute-oriented approach for knowledge discovery in databases. Our approach applies an attribute-oriented concept tree ascension technique in generalization which integrates the machine learning methodology with set-oriented database operations and extracts generalized data from actual data in databases. Our method substantially reduces the computational complexity of the database learning processes. Different knowledge rules, including characteristic rules, discrimination rules, quantitative rules, and data evolution regularities can be discovered efficiently using the attribute-oriented approach. In addition to learning in relational databases, the approach can be applied to knowledge discovery in nexted-relational and deductive databases. The discovered rules can be used in intelligent query answering and semantic query optimization.

Based upon the attribute-oriented induction technique, 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 either the Oracle or SyBase DBMS software. A database learning language for **DBLEARN** is specified in an extended BNF grammar. The system, **DBLEARN**, takes a learning request as input, applies the knowledge discovery algorithm(s) developed in this paper on the data stored in the database, with the assistance of the conceptual hierarchy information stored in the conceptual hierarchy base. Knowledge rules extracted from the database are the output of the learning system. Our primitive experimentation on medium sized data sets demonstrates the effectiveness and efficiency of our methods in knoweldge discovery in relational databases.

Attribute-oriented induction represents a promising direction to follow in the development of efficient learning algorithms for knowledge discovery in databases. There are many issues which should be studied further. The automatic discovery of concept hierarchies in databases, the construction of interactive learning systems, the developement of efficient induction algorithms for object-oriented databases, and the integration of attribute-oriented induction with other learning paradigms are interesting topics for future research.

References

1. Y. Cai, N. Cercone and J. Han, Attribute-Oriented Induction in Relational Databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases, AAAI/MIT Press, 1991, 213–228.*

2. U. S. Chakravarthy, J. Grant and J. Minker, Logic-Based Approach to Semantic Query Optimization, *ACM Trans. Database Syst.*, **15**(2), 1990, 162–207.

3. K. C. C. Chan and A. K. C. Wong, A Statistical Technique for Extracting Classificatory Knowledge from Databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases, AAAI/MIT Press, 1991, 107–124.*

4. F. Cuppens and R. Demolombe, Cooperative Answering: A Methodology to Provide Intelligent Access to Databases, *Proc. 2nd Int. Conf. Expert Database Systems*, Fairfax, VA, April 1988, 621–643.

5. T. G. Dietterich and R. S. Michalski, A Comparative Review of Selected Methods for Learning from Examples, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 41–82.*

6. D. Fisher, Improving Inference Through Conceptual Clustering, *Proc. 1987 AAAI Conf.*, Seattle, Washington, July 1987, 461–465.

7. W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus, Knowledge Discovery in Databases: An Overview, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases, AAAI/MIT Press, 1991, 1–27.*

8. H. Gallaire, J. Minker and J. Nicolas, Logic and Databases: A Deductive Approach, *ACM Comput. Surv.*, **16**(2), 1984, 153–185.

9. M. Genesereth and N. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, 1987.

10. J. Han, Y. Cai and N. Cercone, Data-Driven Discovery of Quantitative Rules in Relational Databases, *IEEE Trans. Knowledge and Data Engineering*, **4**(3), 1992.

11. D. Haussler, Bias, Version Spaces and Valiant's Learning Framework, *Proc. 4th Int. Workshop on Machine Learning*, Irvine, CA, 1987, 324–336.

12. 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 (eds.), *Knowledge Discovery in Databases, AAAI/MIT Press, 1991, 449–462.*

13. M. V. Manago and Y. Kodratoff, Noise and Knowledge Acquisition, *Proc. 10th Int. Joint Conf. Artificial Intelligence*, Milan, Italy, 1987, 348–354.

14. R. S. Michalski, A Theory and Methodology of Inductive Learning, *in Michalski et. al. (eds.), Machine Learning: An Artificial Intelligence Approach, Vol. 1, Morgan Kaufmann, 1983, 83–134.*

15. R. S. Michalski, J. G. Carbonell and T. M. Mitchell, *Machine Learning, An Artificial Intelligence Approach, Vol. 2*, Morgan Kaufmann, 1986.

16. T. M. Mitchell, Generalization as Search, *Artificial Intelligence*, **18**, 1982, 203–226.

17. A. Motro and Q. Yuan, Querying Database Knowledge, *Proc. 1990 ACM-SIGMOD Conf. Management of Data*, Atlantic City, NJ, June 1990, 173–183.

18. M. A. Roth, H. F. Korth and A. Silberschatz, Extended Algebra and Calculus for Nested Relational Databases, *ACM Trans. Database Syst.*, **13**(4), 1988, 389–417.

19. C. D. Shum and R. Muntz, Implicit Representation for Extensional Answers, *Proc. 2nd Int. Conf. Expert Database Systems*, Vienna, VA, April 1988, 497–522.

20. A. Silberschatz, M. Stonebraker and J. D. Ullman, Database Systems: Achievements and Opportunities, *Comm. ACM*, **34**(10), 1991, 94–109.

21. D. Subramanian and J. Feigenbaum, Factorization in Experiment Generation, *Proc. 1986 AAAI Conf.*, Philadelphia, PA, August 1986, 518–522.

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

23. J. Zytkow and J. Baker, Interactive Mining of Regularities in Databases, in G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in Databases, AAAI/MIT Press, 1991, 31–54.*