Extracting Database Knowledge from Query Trees

Jongpil Yoon

Abstract

Although knowledge discovery is increasingly important in databases, the discovered knowledge sets may not be effectively used for application domains. It is partly because knowledge discovery does not take user's interests into account, and too many knowledge sets are discovered to handle efficiently. We believe that user's interests are conveyed by a query and if a nested query is concerned it may include a user's thought process. This paper describes a novel concept for discovering knowledge sets based on query processing. Knowledge discovery process is performed by: extracting features from databases, spanning features to generate range features, and constituting a knowledge set. The contributions of this paper include the following: (1) not only simple queries but also nested queries are considered to discover knowledge sets regarding user's interests and user's thought process, (2) not only positive examples (answer to a query) but also negative examples are considered to discover knowledge sets regarding database abstraction and database exceptions, and (3) finally, the discovered knowledge sets are quantified.

I. Introduction

Recently, a number of studies [3, 4, 6, 9, 10, 12, 19, 21, 23] have concentrated on developing methods of knowledge discovery from databases. These methods are geared towards the representation of large volume of data sets, collected from real-world systems. An excellent survey of the work as of early 1990 appeared in [8, 11, 5]. From an overall analysis of the previous research, we conclude that a viable and efficient strategy for knowledge discovery in large databases would provide the following properties:

Relevance: Knowledge sets are used back against an application database. Not so many knowledge sets, but some of them related and useful to users are discovered.

Richness: Knowledge sets represent not only user's interests but also a user's thought process. Knowledge discovery is performed based on query processing. We believe that user's interests are conveyed by a query and if a nested query is concerned it may include a user's thought process.

Supportiveness: Discovered knowledge sets are more applicable to a database. In order to make a discovered knowledge set more supportive, the knowledge set will be spanned without loosing consistency.

Although knowledge discovery is increasingly important in large databases discovered knowledge sets may not fit user's interests. This is partly because the user's interests are not taken into account. For example, suppose a database includes the two tables, "supply" and "shipment." Let *X1*, *X2* and *X3* be, respectively, the discovered knowledge sets: "the items shipped by agent *A* are taller than 5 feet," "the items shipped by agent *A* are fragile," and "the items shipped to country *C* are fragile." This discovered knowledge set does not always interest to users, especially for someone who poses the query: "list the fragileness of items shipped to country *C*." In this case, only *X3* is useful because this query asks for both fragileness and country. Notice that *X2* should not be excluded if a nested query is posed:"list the fragileness of items shipped to all the countries to which the agent *A* ships."

The key issues addressed in this paper include:

Discovering the "optimal" number of knowledge sets. To reduce the number of discovered knowledge sets, typically, intelligent search strategies are used to trim the search space [11]. Such strategies may not be useful to

Manuscript received October 10, 1995; accepted April 8, 1996.

The author is with Department of Computer Science, Sookmyung Women's University.

This research was supported in part by the Ministry of Information and Communications Fundamental University Research Project.

obtaining a smaller number of knowledge sets. We reduce a search space by focusing on a user's query.

Forming knowledge sets in accordance with user's intentions.

The user's intentions have not been used to discover new knowledge sets. Unless they are used, the discovered knowledge sets may not be useful for the user's application domains. It is further likely that building a nested query means representing user's thought process. While processing a nested query, at an appropriate step knowledge sets can be discovered. Then these sets may represent the steps of user's thought process.

Spanning numeric knowledge sets. Numeric knowledge sets are discovered from databases. Spanning a knowledge set to cover a wider range of data sets is a type of generalization. The spanning method presented in this paper results in increasing "support" while preserving 100% "confidence." Typical approaches (e.g., [2]) are static for both "support" and "confidence," or at least reduce one or the other.

The contributions of this work are as follows: (1) using a user's query, the knowledge sets closely related to the user's interest (query) are discovered; (2) along with query trees, new knowledge sets are dynamically discovered to take a user's thought process into account; (3) the knowledge sets are discovered from the negative examples to be used as integrity constraints; (4) the knowledge sets are spanned to cover a wider range of a database; and (5) the discovered knowledge sets are quantified.

The organization of this paper is as follows: Section 2 investigates the related work. Section 3 reviews the background knowledge. Section 4 provides intuition for our approach. Section 5 describes our method for discovering knowledge sets. An algorithm of knowledge discovery is also presented. In Section 6, we extend the algorithm to consider nested queries. Finally, the contributions of this paper and future work are described in Section 7.

II. Related Work

Knowledge discovery is originated from finding functional dependencies in a database. *Functional dependencies* can be constraints on the set of legal database relationships among data items. Zytkow and Zembowicz have described a way of discovering *regularities* from databases, in the form of a pattern in the range in which that pattern holds [25]. By counting the number of records that have the particular values of attributes, data sets can be grouped and populated in a table.

Until recently, the only methodology available for reasoning about a database has been based on statistical methods [4, 20, 18]. For example, Smyth and Goodman[20] have introduced the form of a probabilistic rule *"IF Y = y then X = x with probability p,"* in which probability $p(x \mid y)$ is added to the rule *"IF Y = y then X = x."* Chan and Wong have used the notion of certainty factor [4]. If $p(object \in C \mid a = A)$ is significantly different from $p(object \in C)$, it is likely that an attribute value A is considered as providing important information for determining whether an object it characterizes should be assigned to a class C.

The Bacon system uses a data analysis algorithm to discover mathematical relationships in numerical data [13, 24]. For example, it rediscovers relationships such as Ohm's law for electric circuits and Archimede's law of displacement.

Agrawal and et al. obtain the classification function using a *training data set* based on a decision tree [1]. The classification function, an interval, is generated by traversing in the decision (or classification) tree from the root to the leaves which are labeled with a particular group label. An example of the classification function is a disjunction of feature vectors: $[(age < 30) \wedge (education \in [0..1]) \vee (age > 30) \wedge (education \in [0..3])]$. The terms "support" and "confidence" are used. Knowledge sets are formed according to the ratio of the frequency of the data set to the total frequency at the value of the attribute for a given threshold. The weakness, however, is that the threshold should be pre-defined and is not adjustable along the level of a decision tree.

Furthermore, discovered knowledge sets are improved, for example, by generalization or specialization. Rules are generalized by using pre-set hierarchical information [3]. This approach using the "domain" dependent hierarchical information disallows improvement of rules in evolutionary systems. It is impossible to pre-set all the domain dependent hierarchical information in a learning system. Rather, we extend the notion of rule generalization developed by Michalski [14]. J. Han, Y. Cai, and N. Cercone have characterized a database in rules which are, then, generalized by using pre-set is_a hierarchy of attribute values [9]. For example, the rule $[\forall class(x) \in freshman \vee class(x) \in sophomore \vee class(x) \in junior \vee class(x) \in senior \rightarrow student(x)]$ can be generalized to $[\forall class(x) \in undergraduate \rightarrow student(x)]$ if *undergraduate* consists of *freshman, sophomore, junior,* and *senior*. It means that all undergraduates are students. This generalization requires the pre-set *is_a hierarchy* about domain values. It is difficult, if not impossible, to set the hierarchy of all possible database domain values over the course of database evolution.

It is conceivable that the space of all possible combinations of attribute values and classes is too large to exhaust. To resolve this problem, a practical learning process has employed heuristics to guide discovery of useful rules [

22]. In contrast, we use a query language, by which we believe the user's background knowledge, interests, and intents are conveyed both to initialize the learning process and to reduce the learning space by considering the answer to the query [23].

III. Preliminaries

Knowledge Model

A *predicate* is a boolean function in the form of either $x \Theta X$ or $x \Theta x$, where x is a variable in the form of *table*[***attribute***], X is a constant, and denotes a comparison connector(>, =, ≠, ≥, etc). A disjunction or a conjunction of predicates about numeric data is defined as a range. For example, for the feature of the weight of supplies, (**supply**[**weight**] ≥ 7) ∧ (**supply**[**weight**] ≤ 9) represents a range for the weights of the table "supply." A *clause* is a disjunction of positive predicates or negative predicates. $L_1 \wedge L_2 \wedge ... \wedge L_i \rightarrow L_{i+1} \vee ... \vee L_n$,
where L_i is a predicate. *A* predicate may be a built-in predicate (aggregate function) such as MAX or MIN. For example, $s,t \in supply$, $(s[weight] = 10) \rightarrow (s[height] > MAX(t[height]))$ specifies that any supply whose weight is 10 are taller than all other supplies.

Positive/Negative Examples

A database query specifies a condition, conveys the user's interests to a database, and its answer is composed of a subset of the database. As a query conveys the user's intention and interests, so does the answer to the query. For example, in an SQL, a query specification is of the form "SELECT attributes FROM table names WHERE a condition." In query processing, the tables in the FROM part of a query are joined if necessary. Query processing yields a resulting table, which is an answer, call *"positive examples."* The tuples of those tables which are specified in the FROM part but not included in the answer, are called *"negative examples."* This notion is very important in that knowledge sets are discovered not only from the positive examples but also from the negative examples. We use the definition in **[23]**.

Definition 1 (Positive and Negative Examples). Consider a set of tables $T = \{R_1,R_2,..,R_n\}$, where R_i is a set of instances. Let A be the answer to a query, a set of the instances resulting from query processing among the tables $R(\subseteq T) = \{R_1,R_2,...,R_m\}$, where $m \leq n$. The instances in A satisfy the query condition q of the query: $A = \sigma_q(R_1 \times R_2 ... \times R_m)[attri_A]$, where σ_q denotes a selection of the instances satisfying q, $R_i \times R_j$ denotes the Cartesian product between R_i and R_j, and $[attri_A]$ denotes a projection of the attributes from the table *A*. We call *A positive examples*. The *negative examples A* is defined as a set of tables, $\bar{A} = \{\bar{A}_1,\bar{A}_2,...,\bar{A}_m\}$, such that A_i is a subset of R_i which does not include the instances included in *A*. That is, $\bar{A}_i = R_i - (A \bowtie R_i)[attri_{Ri}]$, where $R_i \bowtie R_j$ denotes "join" operation between R_i and R_j.

For example, consider a database table

supply

item_no	height	weight	fragileness
I1	25	8	soft
I2	25	7	fragile
I3	30	8	fragile
I4	30	7	fragile
I5	35	8	soft
I6	35	7	fragile

For the query below left, as selecting item_no and weight of the table supply for its weight greater than or equal to 8, the positive and negative examples are:

```
SELECT item_no,
weight
FROM supply
WHERE weight >=8
```

positive-examples

item_no	weight
i1	8
i3	8
i5	8

negative-examples

item_no	height	weight	fragileness
i2	25	7	fragile
i4	30	7	fragile
i6	35	7	fragile

The Nested Query Tree

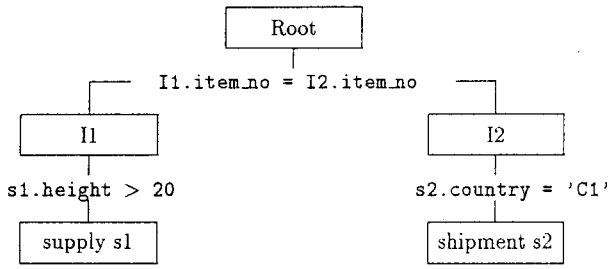
We define a *nested query tree* as a tree structure that corresponds to a nested query expression. In contrast to a typical query tree which corresponds to a relational algebra expression, the query tree used in this paper corresponds to a relational calculus expression. It consists of nodes, each of which represents a table or a condition, and arcs which links between a parent node and a child node. The nodes representing tables can be one of the following: (1) the leaf nodes of the tree represent the base tables which are stored in the database, (2) the internal nodes represent the intermediate tables which result from a sub-query expression, and (3) the root represents the answer to a query.

For example, consider the following query:

```
SELECT   DISTINCT s1.item_no
FROM     supply s1, shipment s2
WHERE    s1.height > 20 AND s2.country = 'C1'
         AND s1.item_no = s2.item_no
```

The query tree of the above nested query is depicted as follows. The leaf nodes "supply s1 and shipment s2" are the tables which are stored in the database. The internal nodes "I1" and "shipment I2" contain the tuples selected from each

table. The root is the answer to the above query.



Confidence and Support

A knowledge piece $s[A] \rightarrow t[B]$ has 90% *confidence* if for those tuples s satisfying $s[A]$, 90% of them also satisfy $t[B]$. All knowledge sets discovered in this paper have 100% confidence.

A knowledge piece $s[A] \rightarrow t[B]$ has 90% *support* if 90% of the all database tuples satisfy the rule. It is denoted as $s[A] \rightarrow t[B] \mid \{90\%\}$. In the same way, a feature $s[X]$ discovered from positive examples can have $n\%$ support if $n\%$ of the positive examples satisfy the feature $s[X]$, and it is denoted as $s[X]|\{n\%\}$. Similarly, the support percentage for features discovered from negative examples are obtained based on the negative examples. To find the support for each discovered knowledge, we use a frequency graph. The frequency graph will be explained in the following section.

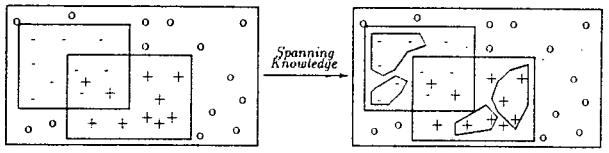


Fig. 1. Spanning positive and negative examples.

IV. Intuition

In general, a database consists of one or more tables and a query is posed to some of those tables by specifying in the **FROM** part. Figure 1 shows spanning knowledge sets for both positive examples and negative examples in a database. The left picture (a) depicts a database containing the tuples of those tables that are specified in the **FROM** part. Some tuples satisfy the query and some do not. Those tuples satisfying the query are regarded as positive examples (denoted as + sign), while those not satisfying the query are regarded as negative examples (denoted as - sign). Notice that "**O**" signs denote tuples in the tables which are not involved in a give query. For an attribute, some values in the tuples appear in the upper left box, some appear in the lower middle box, and some appear in the intersection. The upper left box contains negative examples, the lower middle box contains positive examples, and the intersection of the two boxes contains mixed examples for a particular attribute.

If these examples are numeric values, those adjacent values may be spanned unless causing inconsistency. (The inconsistency will be discussed in the following section.) The picture(b), therefore, depicts the same database, as the picture (a) does, containing the positive and negative values spanned with neighbors.

For example, (**s[weight]** = 8) represents the *feature* such that the weight of a supply **s** is 8. In this paper, we span features to generate a range feature. In order to span features, the data must be sorted. We first sort the data in terms of attributes. For each attribute, the frequency of those values is considered as depicted in a frequency graph. The features extracted from either positive or negative examples can be spanned. For example in Figure 2, denoted as the *frequency graph* of the attribute v, consider each feature: $(v = V1)|\{20\%\}$, $(v = V2)|\{27\%\}$, ...,$(v = V10)|\{18\%\}$. From the first two features, $(v = V1)$ and $(v = V2)$, if no negative example appears in between, the two features are spanned to be a range feature $(v \geq V1) \wedge (v \leq V2)|\{47\%\}$. Furthermore with $(v = V3)$, the range feature is spanned to be a wider range $(v \geq V1) \wedge (v \leq V3)|\{60\%\}$. In the same manner, the spanning knowledge sets are as follows: $(v \geq V7) \wedge (v \leq V8)|\{27\%\}$ for the positive examples, and $(v \geq V4) \wedge (v \leq V5)|\{45\%\}$ for the negative examples.

V. Knowledge Discovery on Query Processing

In our approach, knowledge discovery is performed when a query is posed by a user (or recommendably by an expert like a **DBA**). For the query, the answer is generated. New features may be extracted from positive examples (or the answer) and negative examples, both of which are a subset of the database. Extracted features are then spanned with the neighbors. Spanned features are associated with a query condition to constitute a knowledge set. The steps of discovering knowledge sets are as follows:

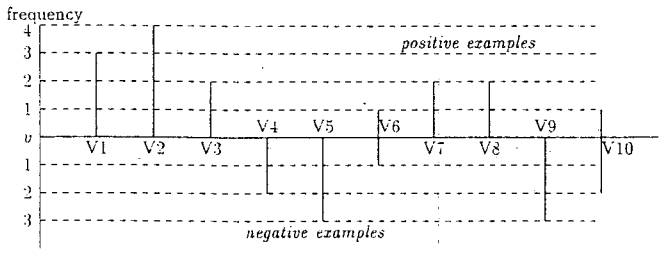
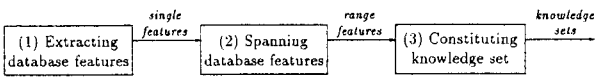


Fig. 2. Positive and negative examples in a frequency graph.



Each step will be discussed in the subsections in great detail.

1. Extracting Database Features

The feature sets should be satisfied by all the positive examples but not be satisfied by any negative examples [15, 23].

Definition 2 (Extracted Features). Consider a table T. Let T_f be a set of tuples that is the answer to a query Q, and T_f be $T - T_t$. A feature X can be extracted from T such that $\forall t \in T_t$, $t' \in T_f$, $(t \models X) \wedge \neg(t' \models X)$. □

For example, given a query, the values of an attribute v in the answer are $V1, V2, V3, V6, V7, V8, V10, V1, V2, V3, V7, V8, V1, V2, V2$, and those in the negative examples are $V4, V5, V6, V9, V10, V4, V5, V9, V10, V5, V9$. Suppose that these examples are in an order and depicted as in Figure 2. Especially for numerical data, those values as above can be grouped and sorted very easily. It is clear that the feature $(v = V1)$ is extracted from the positive examples. No feature, however, is extracted for $v = V6$ according to Definition 2 because the values for that attribute are included not only in positive examples but also in negative examples. From these examples, the following features are simply extracted: $(v = V1)$, $(v = V2)$, $(v = V3)$, $(v = V7)$, and $(v = V8)$ from the positive examples, and $(v = V4)$, $(v = V5)$, and $(v = V9)$ from the negative examples.

2. Spanning Database Features

Those features sets extracted as in the previous section can be spanned when data are sorted as shown in Figure 2. This spanning step is performed based on Definition 2. The feature sets, each feature is called *single feature*, extracted from the positive answers are spanned with the adjacent positive examples to become so called *range features* as far as no negative examples appear in between. Likewise, the features extracted from negative examples also continue to be spanned as far as no positive examples appear.

Theorem 1 (Spanning Database Features). Suppose either a single feature $(v = V_i)$ or a range feature $(v \geq V_i) \wedge (v \leq V_k)$ are extracted from positive examples. If a single feature$(v = V_j)$ is extracted from positive examples, and there is no V' in negative examples such that $V_i \geq V' \leq V_j$, then $(v \geq V_i) \wedge (v \leq V_j)$ holds, and vice versa. □

Proof: Given a database T, and let T_t and T_f be positive and negative examples respectively, and q be the query condition. Suppose that a feature $(v \geq V_i) \wedge (v \leq V_j)$ holds, and there exists V' in T_f such that $V_i \geq V' \leq V_j$. Then, according to Definition 2, for all t in T_t, any V' between V_i and V_j, $(v = V')$ should be satisfied with t. That is, $(v = V')$ is extracted from T_t, which is contrary to the assumption V' is in T_f. □

For example in Figure 2, the features extracted from positive examples, $(v = V1)$ and $(v = V2)$, are spanned to be a range feature, $(v \geq V1) \wedge (v \leq V2)$, because no negative examples appear. This range feature then continue to be spanned with the feature $(v = V3)$ to become a new range feature $(v \geq V1) \wedge (v \leq V3)$. However, this range feature cannot be spanned with the feature $(v = V4)$ because of being extracted from negative examples. In the same manner, the range features, $(v \geq V4) \wedge (v \leq V5)$ and $(v \geq V7) \wedge (v \leq V8)$ are obtained for negative examples and positive examples, respectively.

3. Constituting Knowledge Sets

Given the range features or the single features, knowledge sets will be constituted in accordance with a query. Recall that as the features are obtained based on a query, so do knowledge sets.

Theorem 2 (Constituting Knowledge Sets). If a feature e (either a single feature or a range feature) is extracted from positive examples and further spanned, and if those positive examples are obtained by answering a query q, then $e \rightarrow q$ holds. On the other hand, if e is extracted from negative examples and further spanned, and if those negative examples are obtained by answering the q, then $e \rightarrow \neg q$ holds. □

Proof: Given a database T, and let T_t and T_f be positive and negative examples respectively, and q be the query condition. Since e is extracted from T_t, $e \supseteq T_t$. It is also known that $T_t \models q$. Therefore, $e \rightarrow q$. On the other hand, If e is extracted from T_f, e $\supseteq T_f$. It is also known that $T_f \models \neg q$. Therefore, e $\rightarrow \neg q$.

The features e extracted (or further spanned) from positive examples can be used to constitute a knowledge set in the form of parts. With respect to a query q, based on which the positive examples are obtained, a discovered knowledge set will be the form: $e \rightarrow q$. In the same way, for the features e extracted (or further spanned) from negative examples, a discovered knowledge set will be: $e \rightarrow \neg q$.

The three steps we have developed so far appear in the algorithm as in Figure 3. We observe that using the *quick sort* for the loop in #1 takes the average time complexity of $O(nlogn)$, where n is the number of tuples. The loop of #6 or #7 takes only $O(m^2)$, where m is the number of attributes, and $m \ll n$ in databases.

EXAMPLE 5.1

Suppose the following query is posed to the database shown in Figure 4.

SELECT s1.item_no, s1.fragileness, s1.weight, s2.quantity
FROM supply s1, shipment s2
WHERE s2.country = 'C1' **AND** s1.item_no = s2.item_no

According to the algorithm in Figure 3, first, the given table is divided into positive and negative examples. Those two examples are in Figure 5.

Algorithm:
Input: An SQL-like query.
Output: A set of rules.
Method: (Assume that an SQL query is presented and the positive examples A and the negative examples
A are generated in the form of table.) /* A and A are generated according to Definition 1. */
Let q be the condition of the SQL query. Let S be a set of attributes $\{a_1,a_2,...,a_i,...,a_n\}$ specified in **SELECT** part of the SQL query. Let n be the number of attributes in S
for each a_i in S **do**/* Let a_i be the "target-attribute" */
begin
1. Sort A and $\bar{A}$ by a_i;
2. Group A and A by a_i;
3. Let V and V denote the set of values of a_i in A and $\bar{A}$, respectively;
   Let $|V|$ and $|V|$ be the number of distinctive values (the so called size of) in A and, $\bar{A}$ respectively.
4. $k := 1$;
5. $W := \{\}$; Let Wi be the nth element of the set W and $|W|$ be the size of W;
6. **do until** k $|V|$ **begin**
   (a) Add V_k into the set W; /* append V_k to W */
   (b) if $V_{k+1} \in \bar{A}$ or $k \geq |V|$ then begin
      (1) $m := |W|$; /* let m be the size of W */
      (2) **if** $m = 1$ **then** the rule is $(a_i = W_l) \rightarrow q$
         else the rule is $(a_i \geq W_l) \wedge (a_i \leq W_m) \rightarrow q$;
      (3) $W := \{\}$;
   end
   (c) $k := k + 1$;
   end;
7. $l := 1$;
8. **do until** $l \leq |\nabla|$ **begin**
   (a) Add V_l into the set W; /* append V_k to W */
   (b) **if** $V_{l+1} \in A$ or $l \geq |\nabla|$ **then begin**
      (1) $m := |W|$;
      (2) **if** $m = 1$ **then** the rule is $(a_i = W_l) \rightarrow \neg q$
         else the rule is $(a_i \geq W_l) \wedge (a_i \leq W_m) \rightarrow \neg q$;
      (3) $W := \{\}$;
   end
   (c) $l := l + 1$;
   end;
end.

Fig. 3. Algorithm of numerical knowledge discovery.

supply

item_no	height	weight	fragileness
i1	25	8	soft
i2	25	7	fragile
i3	30	6	fragile
i4	30	4	fragile
i5	45	8	soft
i6	45	10	fragile
i7	35	8	fragile

shipment

agent	item_no	country	quantity
A1	i4	c1	22
A1	i1	c2	12
A1	i2	c1	22
A2	i2	c2	42
A2	i3	c1	17
A2	i5	c3	29
A3	i8	c4	22
A3	i7	c2	18

Fig. 4. Supply and shipment Example.

Positive Examples /* the result of join operation*/

item_no	fragileness	wight	quantity
i2	fragile	7	22
i3	fragile	6	17
i4	fragile	4	22

Negative Examples

item_no	height	weight	fragileness
i1	24	8	soft
i5	45	8	soft
i6	45	10	fragile
i7	35	8	fragile

Negative Examples

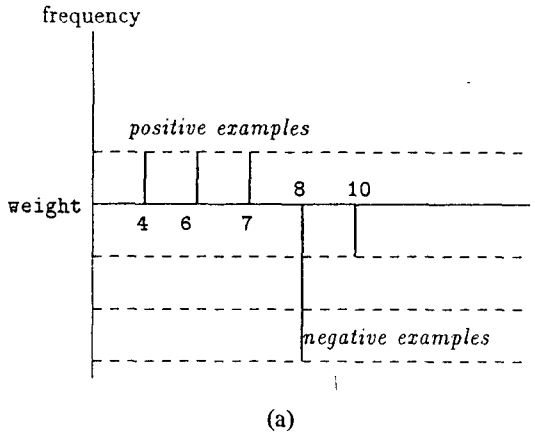
agent	item-no	country	quantity
A1	i1	c2	12
A2	i2	c2	42
A2	i5	c2	29
A3	i8	c4	22
A3	i7	c2	18

Fig. 5. Positive and Negative Examples of the Network Example.

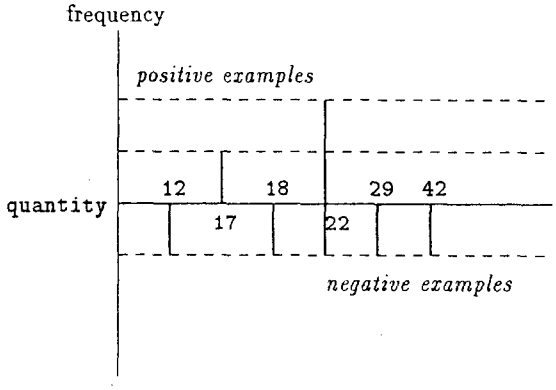
The values of those attributes specified in the **SELECT** part in the given query are sorted and their frequencies are depicted as in Figure 6. In the figure, the picture (a) represents the frequency of the **weight** values, whereas (b) represents for the **quantity** values. We will consider the positive or the negative examples for each attribute.

Simple features in the form of "(attribute = value)" pair can be spanned to be range features. For example, in Figure 6(a), those features extracted from positive examples are (*weight* = 4), (*weight* = 6), and (*weight* = 7), and those extracted from negative examples are (*weight* = 8) and (*weight* = 10). The feature (*weight* = 4) is spanned with (*weight* = 6) to generate the range feature (*weight* $\geq$ 4) $\wedge$ (*weight* $\leq$ 6), without causing the inconsistency, the inconsistency such that negative examples appear within the spanned range feature. Spanning continues to form the range (*weight* $\geq$ 4) $\wedge$ (*weight* $\leq$ 7). Similarly, for negative

examples, the features (*weight* = 8) and (*weight* = 10) are spanned to be a range feature (*weight* ≥ 8) ∧ (*weight* ≤ 10).



(a)



(b)

Fig. 6. Positive/negative knowledge spanning-example

Knowing the above features, by considering the query, the following knowledge sets can be constituted for the attribute "weight."

r ∈ *supply*, *s* ∈ *shipment*, (*r*[*weight*] ≥ 4) ∧ (*r*[*weight*] ≤ 7) → (*s*[*country*] = "C1")|{100%}

r ∈ *supply*, *s* ∈ *shipment*, (*r*[*weight*] ≥ 8) ∧ (*r*[*weight*] ≤ 10) → ¬(*s*[*country*] = "C1")|{100%}

The first knowledge specifies that a supply whose weight is between 4 and 7 is shipped to the country "C1" with 100% support, while the second specifies that a supply whose weight is between 8 and 10 is not shipped to the country "C1" with 100% support. Similarly, from (b) of Figure 6, the following knowledge sets can be discovered with respect to the attribute "quantity."

r ∈ *supply*, *s* ∈ *shipment*, (*r*[*quantity*] = 17) → (*s*[*country*] = "C1")|{33%}

r ∈ *supply*, *s* ∈ *shipment*, (*r*[*quantity*] = 18) → ¬(*s*[*country*] = "C1")|{20%}

r 2 *supply*; *s* ∈ *shipment*; (*r*[*quantity*] = 12) → ¬(*s*[*country*] = "C1")|{20%}

r 2 *supply*; *s* 2 *shipment*; (*r*[*quantity*] 29) ∧ (*r*[*quantity*] ≤ 42) → ¬(*s*[*country*] = "C1")|{40%}

Notice that for quantity of 22, no feature will be generated because of inconsistency according to Definition 2. The first knowledge is interpreted that a shipment whose quantity is 17 is shipped to the country "C1." The rests are interpreted in the same way and omitted. □

The knowledge sets discovered from the positive examples specify the dependencies among attributes. The knowledge sets discovered from the negative examples specify the exceptions in a database, and so they can be used as "integrity constraints" in the database.

VI. Knowledge Discovery via Nested Queries

In this section, we will extend the basic concepts described in the previous section toward nested queries. Nested queries are specified in a more complicated way and convey not only user's intention but also user's "thought process." It is likely that the query trees for nested queries will be more complicated and the answers will retain more meanings as well. We believe that the query answering process reflects the user's thought process. We consider the following two types of nested queries. (1) queries for membership comparisons and (2) queries for set comparisons. Knowledge discovery is carried out on a nested query tree. Not only the root node (for an answer), but also each internal node of a query tree will be the discovery space. These knowledge sets may be also useful if the user's intention is not just at the root node of the nested query tree.

1. Queries for Membership Comparisons

A nested query may have a membership comparison within the WHERE part. The membership comparator includes: >ALL, >SOME, <ALL, <SOME, IN. This section describes how the membership comparison used in a query is taken into account. The steps of using a nested query for knowledge discovery:

1. construct a query tree (as discussed in Section 3)
2. traverse from leaf nodes
3. let its level be *l*
4. apply the algorithm in Figure 3 onto the node in the level (*l* - 1)

(a) discover a knowledge set and set *l* = *l* - 1

(b) if there exists a membership comparison then constitute the knowledge set with an RHS associate from Table1

5. repeat the steps 3 to 4 until ($l \geq 1$)

For example, consider the following query specified to ask for the attributes "item_no" and "weight" for those supplies which are taller than any items which are shipped to the country C1.

Table 1. Membership Comparisons.

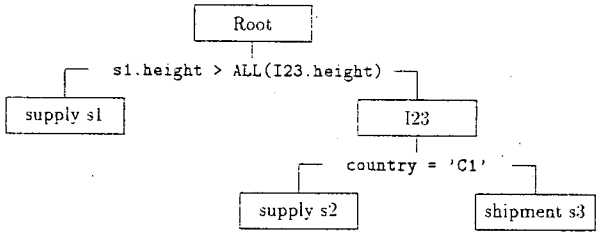
set-element comparison	meaning	RHS(q) associates
$\alpha \geq$ALL A	α is greater than or equal to MAX of A	($\alpha \geq$MAX(A))
$\alpha \geq$SOME A	α is greater then or equal to MIN of A	($\alpha \geq$MIN(A))
$\alpha \leq$ALL A	α is less than or equal to MAX of A	($\alpha \leq$MAX(A))
$\alpha \leq$SOME A	α is less than or equal to MIN of A	($\alpha \leq$MIN(A))
$\alpha >$ALL A	α is greater than MAX of A	($\alpha >$MAX(A))
$\alpha >$SOME A	α is greater than MIN of A	($\alpha >$MIN(A))
$\alpha <$ALL A	α is less than MAX of A	($\alpha <$MAX(A))
$\alpha <$ SOME A	α is less than MIN of A	($\alpha <$MIN(A))
α IN A	α is IN A	($\alpha \in$(A))

Note that α denotes a set of attributes, A denotes a set of the value sets for each attribute in α, and q denotes the query condition shown in the root of the query trees. In $\alpha \ominus$A, the comparison connector $\ominus$ is applied to an attribute in the α set with the corresponding value set in the A set

Suppose this query is posed to the database containing the tables shown in Figure 4.

```
SELECT  s1.item_no, s1.weight
FROM    supply s1
WHERE   s1.height >
          ALL(SELECT s2.height
                FROM supply s2, shipment s3
                WHERE s3.country = 'C1')
                      AND s2.item_no = s3.item_no
```

The query tree for the above query is as follows:



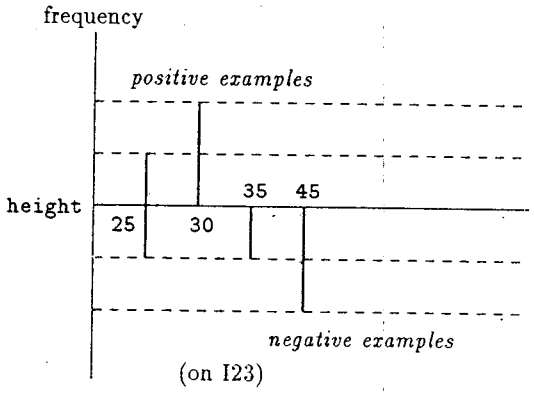
The positive and negative examples in terms of the above query are depicted in Figure 7.

Notice that spanning regarding the same attribute is different depending on queries. It is because that the values for "weight" are different between Figure 6 and Figure 7.

By applying the algorithm in Figure 3 to the internal node "I23" in the level 2 of Figure 7, the following knowledge set is formulated.

r ∈ supply, s ∈ shipment, (r[height] = 30) → (s[country] = "C1")|{67%}

r ∈ supply, s ∈ shipment, (r[height] ≥ 35) ∧ (r[height] ≤ 45) → ¬(s[country] = "C1")|{75%}



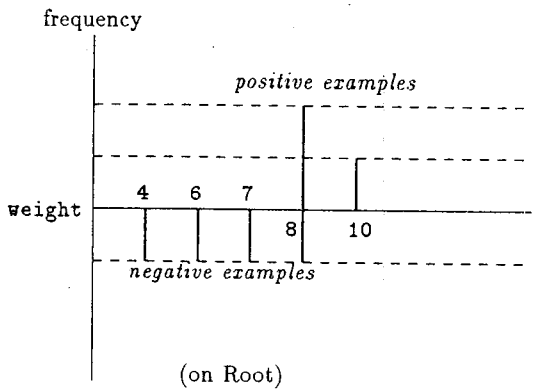


Fig. 7. Knowledge spanning on the set-membership query.

The first knowledge specifies that the items are shipped to the country "C1" if their heights is 30 with 67% of support. The second one specifies that the items whose height is between 35 and 45 are not shipped to the country "C1" with 75% of support. Similarly, from the root node of Figure 7, the following knowledge set can be discovered by taking an RHS associate MAX from Table 1:

r ∈ supply, s ∈ supply, t ∈ shipment; (r[weight] = 10) → (r[height] > MAX(s[height]) ∧ (t[country] = "C1")|{33%}

r ∈ supply, s ∈ supply, t ∈ shipment, (r[weight] ≥ 4) ∧ (r[weight] ≤ 7) → ¬{(r[height] > MAX(s[height]) ∧ (t[country] = "C1"))}|{75%}

The first knowledge describes that those items whose weight is 10 are taller than the maximum height of those items shipped to the country "C1" with 50% of support. The second one describes that the items whose weight is between

4 and 7 do not are taller than the maximum height of those items shipped to the country "C1" with 60% of support. □

2. Queries for Set Comparisons

In a nested query, we may want to compare intermediate two sets (i.e., two intermediate tables) to see if one set contains the other set. Such comparisons are made in an SQL expression using the CONTAINS, NOT CONTAINS, and NOT EXISTS with MINUS constructs. In this section, we introduce how the discovered knowledge sets are quantified.

Consider the following knowledge:

$$\exists s \, \forall t, \; L_i(s) \rightarrow L_j(t)$$

The above is interpreted: if there exists any tuples s in which L_i(s) holds, then for all tuples *t*, L_j(t) holds. Choosing a quantifier between universal ($\forall$) and existential ($\exists$), is determined based on a query expression. A knowledge set discovered from a subquery can be quantified if the subquery associates with such a set comparative construct as exemplified above.

For example, consider the query specified to ask for the attributes "height" and "quantity" of those items shipped by the agents which ship to all the same country as agent "A1" does. Suppose this query is posed to the database containing the tables shown in Figure 4.

```
SELECT s1.height, s2.quantity
FROM   supply s1, shipment s2
WHERE NOT EXISTS
        ( SELECT DISTINCT  s3.country
          FROM   shipment s3
          WHERE s3.agent = 'A1'
          MINUS
          SELECT DISTINCT  s2.country
          FROM   shipment s4
          WHERE s2.agent = s4.agent
                  AND s1.item_no = s2.item_no )
```

The query tree of the above nested query is depicted in Figure 8. The frequency graph with respect to "quantity" on the node "I3" and the one with respect to "height" on the node "I12" are depicted in Figure 9.

By applying the algorithm in Figure 3 to the node "I3" of Figure 9, the following knowledge set is formulated.

r ∈ *shipment*, (*r*[*quantity*] = 12) → (*r*[*agent*] = "A1")|{33%}
r ∈ *shipment*, (*r*[*quantity*] ≥ 17) ∧ (*r*[*quantity*] ≤ 18) → ¬(*r*[*agent*] = "A1")|{40%}
r ∈ *shipment*, (*r*[*quantity*] ≥ 29) ∧ (*r*[*quantity*] ≤ 42) → ¬(*r*[*agent*] = "A1")|{40%}

Similarly, from the node "I12" of Figure 9, the following knowledge set can be discovered:

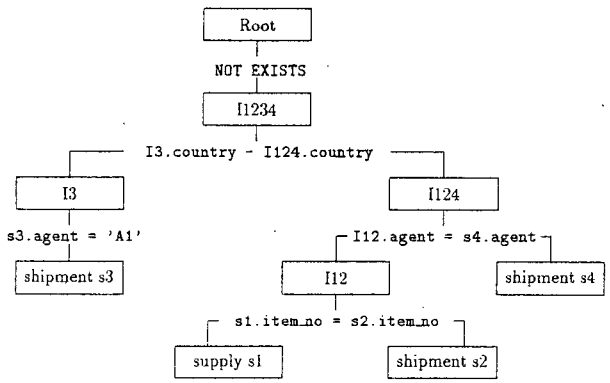


Fig. 8. A query tree fro an example query

s ∈ *supply*, *t* ∈ *shipment*, (*s*[*height*] ≥ 25) ∧ (*s*[*height*] ≤ 35)→ (*s*[*item_no*] = *t*[*item_no*])|{86%}

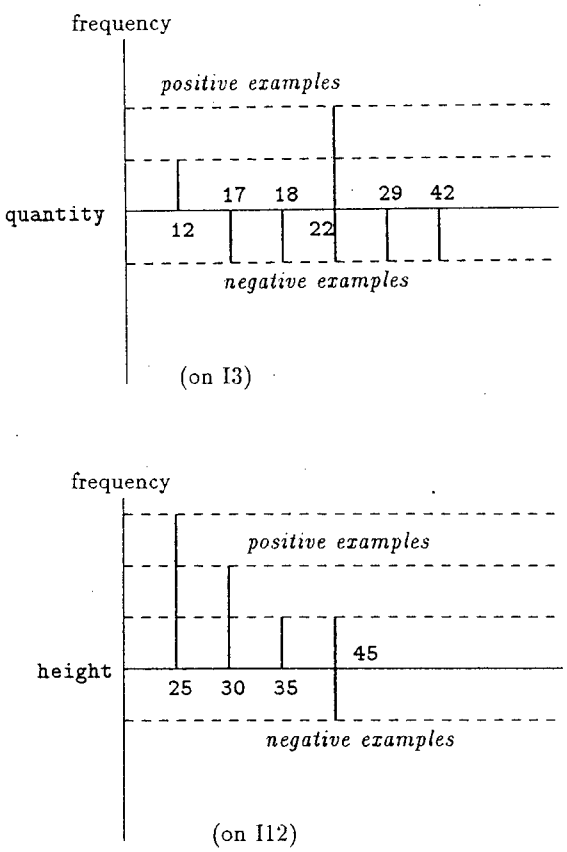


Fig. 9. Knowledge spanning on the set-set comparison query.

The discovered knowledge specifies that those supplies whose height is between 25 and 35 are shipped with 86% of support. Notice that among the supplies whose height is 45,

the item i5 is shipped, while the item i6 is not. We can further traverse to the internal nodes "I124," and "I1234." The positive and negative examples in terms of the query are depicted in Figure 10.

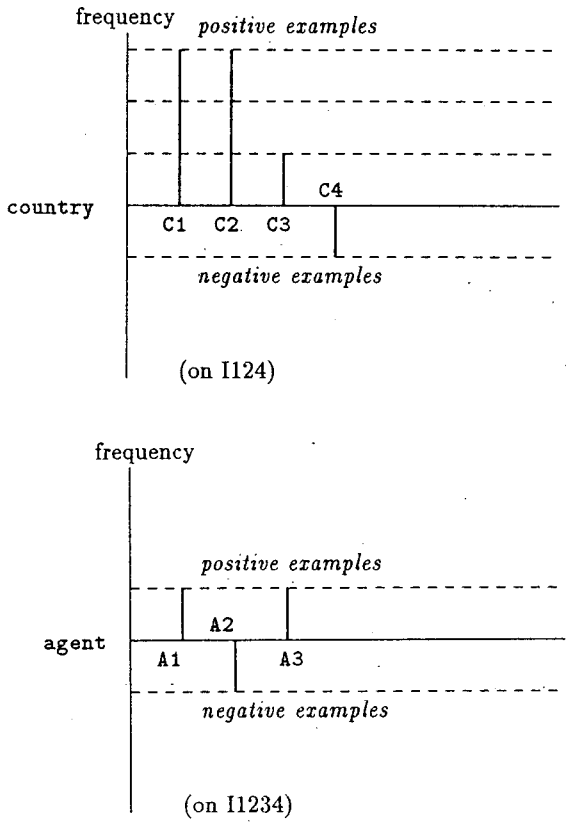


Fig. 10. Knowledge spanning on the set-set comparison query

By applying the algorithm in Figure 3, as in Figure 10, the knowledge set discovered with respect to "country" is:

$\exists r \in supply,\ s \in shipment, \forall t \in shipment;$
$(s[country] = C1) \wedge (s[country] = C2) \wedge (s[country] = C3)$
$\rightarrow (r[item\_no] = s[item\_no]) \wedge (s[agent] = t[agent])|\{100\%\}$
$\exists r \in supply,\ s \in shipment, \forall t \in shipment,\ (s[country] = C4)$
$\rightarrow \neg|\{(r[item\_no] = s[item\_no]) \wedge (s[agent] = t[agent])\}|\{100\%\}$

The first knowledge above specifies that all the items shipped to those countries, C1, C2, and C3, are listed in the supply table. The second one specifies that all the items shipped to the country, C4, are not listed in the supply table.

The knowledge set discovered with respect to "agent" is:

$\exists r \in supply,\ \forall s \in shipment,\ t \in shipment,\ (s[agent] = A1)$
$\wedge (s[agent] = A3) \rightarrow |\{(s[country] = t[country]) \wedge$
$(r[item\_no] = s[item\_no]) \wedge (s[agent] = t[agent]) \wedge (t[agent] = A1)\}|\{100\%\}$
$\exists r \in supply,\ \forall s \in shipment,\ t \in shipment,\ (s[agent] = A2)$
$\rightarrow \neg|\{(s[country] = t[country]) \wedge (r[item\_no] =$
$s[item\_no]) \wedge (s[agent] = t[agent]) \wedge (t[agent] = A1)\}|\{100\%\}$

The first knowledge specifies that the agents, A1, and A3, ship to all the same country as the agent A1 ships. The second one specifies that the agent, A2, does not ship to all the same country as the agent A1 ships.

Finally, consider the root node of the query tree. The positive and negative examples in terms of the query are depicted in Figure 11.

By applying the algorithm in Figure 3 to the node "Root" of Figure 10, some of the newly discovered knowledge set include:

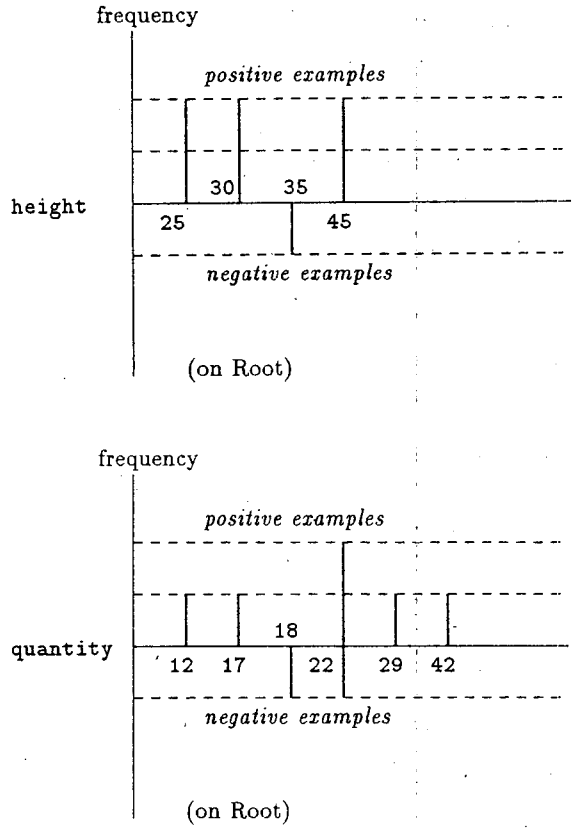


Fig. 11. Knowledge spanning on the set-set comparison query.

$\exists r \in supply,\ \forall s \in shipment,\ t \in shipment,\ (r[height] \geq 25) \wedge$
$(r[height] \leq 30) \rightarrow |\{(s[country] = t[country]) \wedge$
$(r[item\_no] = s[item\_no]) \wedge (s[agent] = t[agent]) \wedge$
$(t[agent] = A1)\}|\{67\%\}$
$\exists r \in supply,\ \forall s \in shipment,\ t \in shipment;\ (r[quantity] \geq 12) \wedge$
$(r[quantity] \leq 17) \rightarrow |\{(s[country] = t[country])$
$\wedge$ (r[item_no] = s[item_no]) $\wedge$ $(s[agent] = t[agent]) \wedge$
$(t[agent] = A1)\}|\{33\%\}$

The first knowledge specifies that those items whose height is between 25 and 30 are shipped by an agent that ships to all the same countries as the agent "A1." The second one specifies that those items shipped out in the quantity of between 12 and 17 are shipped by an agent that ships to all the same countries as the agent "A1."

VII. Conclusion

This paper describes a novel concept for knowledge discovery from databases. Knowledge sets can be discovered based on a user's query posed to a database. Both positive and negative examples are divided according to a user's query, and are used to extract database features.

The features extracted from positive examples are spanned within positive examples to enlarge the percentage of their support. The features extracted from negative examples are spanned as well. Then, a knowledge sets are constituted with the spanned features and the given query. All the knowledge sets discovered by the above method have 100% confidence with maximally enlarged support. We also extend the knowledge discovery method to nested queries so that richer knowledge sets can be discovered.

The benefits described in this paper include: (1) knowledge sets more suitable to the user's interests are discovered because discovery process is initiated by a query which conveys the user's interests; (2) a higher performance for discovery processing results from the smaller search space, by using the positive and negative examples rather than using the entire database; (3) knowledge sets are spanned to cover a wider range of a database, that is, the percentage of their support is maximally enlarged; (4) knowledge sets discovered from negative examples can be used as constraints, which of course represents the exceptions of a database. (5) knowledge sets discovered based on nested queries can represent user's thought process. The knowledge discovery process can be applied to large, real world databases obtained from network fault/alarm data, satellite data, aeronautical data, or meteorological data. This approach may also be useful for research in "cooperative answers" [16, 7],in which the system responds with knowledge sets rather than simply providing the tuple sets. Knowledge sets which may be discovered from the above application domains need to be specified in a concise and elegant form, for example, by generalization techniques.

References

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering*,5(6):914-925, 1993.

[2] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining association rules between sets of items in large databases," In Sushil Jajodia, editor, *Proc. ACM SIGMOD Intl. Conf. on Management of Data,* pages 207-216,Washington D.C., 1993.

[3] Y. Cai, N. Cercone, and J. Han "An attribute-oriented approach for learning classification rules from relational databases," *In the Sixth Int'l Conf. of Data Engineering,* pages 281-288, Los Angeles, Calif, 1990.

[4] K. C. Chan and A. K. Wong, "A statistical technique for extracting classificatory knowledge from databases," In Gregory Piatetsky-Shapiro and William J. Frawley,editors, *Knowledge Discovery in Databases,* pages 107-124. MIT Press, 1991.

[5] V. Fayyad, G. Piatesky-Shapiro, P. Smith, and R. Uthurusany, *Advances in Knowledge Discovery and Data mining*, Edited, AAAI press / The MIT press, 1996.

[6] B. Gaines, "Extracting knowledge from data," *Proc. IJCAI-89 Workshop on Knowledge Discovery in Databases*, pages 109-116, 1989.

[7] Terry Gaasterland, Parke Godfrey, and Jack Minker, "Relaxation as a platform for cooperative answering," *Journal of Intelligent Information Systems,* 1(3-4): 293-321, 1992.

[8] edited Gregory Piatetsky-Shapiro. *Knowledge Discovery in Databases.* MIT Press, MA, 1991.

[9] J. Han, Y. Cai, and N. Cercone, "Data-driven discovery of quantitative rules in relational databases," *IEEE Transactions on Knowledge and Data Engineering*, 5(1):29-40, 1993.

[10] J. Hong and C. Mao," Incremental discovery of rules and structure by hierarchical and parallel clustering," In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 177-194. MIT Press, 1991.

[11] Marcel Holsheimer and Arno Siebes," Data mining: The search for knowledge in databases," Technical Report CS-R9406, ISSN 0169-118X, CWI, 1090 GB Amsterdam, 1993.

[12] K. Kaufman, R. Michalski, and L. Kerschberg, "Mining for knowledge in databases: Goals and general description of the INLEN system," Proc. IJCAI-89 *Workshop on Knowledge Discovery in Databases*, pages 158-172, 1989.

[13] Pat Langley, Gary L. Bradshaw, and Herbert A. Simon, "Rediscovering chemistry with the Bacon system," In Ryszard S. Michalski, Jamine G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning, an Artificial Intelligence approach*, Volume 2, pages 307-329. Morgan Kaufmann, San Mateo, CA,1986.

[14] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell. *Machine Learning*. Morgan Kaufmann, Los Altos, 1983.

[15] R. S. Michalski and J. B. Larson. Incremental generation of VL1 hypotheses: The underlying methodology and the description of the program AQ11. Technical Report UIUCDCS-F-83-905, Univ. of Illinois, Dept. of Computer Science, 1983.

[16] Ami Motro. FLEX: A tolerant and cooperative user interface to database. *IEEE Transactions on Knowledge and Data Engineering*, 2, 1990.

[17] Ryzard S. Michalaski and Gheorghe Tecuci. *the First International Workshop on Multi-strategy Learning*. Harpers Ferry, 1991.

[18] G. Piatetsky-Shapiro and C. J. Matheus. Measuring data dependencies in large databases. In G. Piatetsky-Shapiro, editor, *AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 162-173, Washington, D.C., 1993.

[19] J. R. Quinlan. Generating production rules from decision trees. In *Proc. of Int'l Joint Conf. on Artificial Intelligence*, pages 304-307. 1987.

[20] P. Smyth and R. M. Goodman. Rule induction using information theory. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 159-176. MIT Press, 1991.

[21] P. Smyth and R. M. Goodman. An information theories approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*,(4):301-316, 1992.

[22] B. G. Silverman, M. R. Hieb, and T. M. Mezher. Unsupervised discovery in an operational control setting. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 432-448. MIT Press, 1991.

[23] Jong P. Yoon and Larry Kerschberg. A framework for knowledge discovery and evolution in databases. *IEEE Transactions on Knowledge and Data Engineering*,5(6):973-978, December 1993.

[24] Jan M. Zytkow. Combining many searches in the FAHRENHEIT discovery system. In *the 4th Int'l Workshop on Machine Learning*, pages 281-287, San Mateo, CA,1987.

[25] Jan M. Zytkow and Robert Zembowicz. Database exploration in search of regularities. In *Unpublished manuscript*. 1993.

Jongpil Yoon received PhD degree from George Mason University, Fairfax, Virginia, in 1993, MS degree from University of Florida, Gainesville, Florida, in 1987, and BE degree from Yonsei University, Seoul, in 1981. He is Assistant Professor, Department of Computer Science, Sookmyung Women's University. Dr. Yoon is a member of the ACM SIGMOD and IEEE Computer Society, and has been a reviewer for IEEE Transactions on Knowledge and Data Engineering, Journal of Information Intelligent Systems, and ACM SIGMOD conferences. He is a program committee of International Workshop on Advanced Transaction Models and Architectures in Conjunction with VLDB'96.