PROJECT REPORT

KNOWLEDGE DISCOVERY FROM DATABASE

SUBMITTED IN PARTIAL FULFILMENT OF THE DEGREE OF
BACHELOR OF TECHNOLOGY

by

Debojyoti Kar

Under the guidance of

Mr.K.A.Abdul Nazeer

Department of Computer Engineering
National Institute of Technology, Calicut
National Institute of Technology, Calicut
Department of Computer Engineering

Certified that this Project Report entitled

KNOWLEDGE DISCOVERY FROM DATABASE

is a bonafide report of the projectwork done by

Debojyoti Kar

in partial fulfilment of the degree of
Bachelor of Technology
under our guidance

Mr.K.A.Abdul Nazeer
Project Guide
Senior Lecturer
Dept.of Computer Engineering

Dr.V.K.Govindan
Professor and
Head
Dept.of Computer Engineering
Acknowledgement

I express my sincere gratitude and acknowledge my indebtedness to Mr. K. A. Abdul Nazeer, Senior Lecturer, Department of Computer Science and Engineering, National Institute of Technology, Calicut, for his inspiring guidance and valuable suggestions and the inestimable help he has rendered throughout the work.

I also express my deep sense of gratitude to Mr. Pramod, administrator of Software Research Laboratory, for providing me all the necessary tools and co-operation to make my work a success. Last but not the least, I like to thank all my friends who stood by me and motivated me during various stages of this work.

Debojyoti Kar
Abstract

Knowledge discovery in databases, or data mining, is an important issue in the development of database and knowledge 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 technique, 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 queries and facilitate semantic query optimizations. Based upon these principles, we need to build a natural language interface to the database which will successfully extract knowledge from the database. The need for the interface has become increasingly acute as more and more people access information through their web browsers, PDAs, and cell phones. Natural language questions need to be mapped correctly to SQL queries and then using the attribute-oriented approach knowledge is extracted successfully from the back-end database. The backend database stores a large amount of information-rich, relatively reliable and stable data. The knowledge discovery process is initiated by the user by putting forward questions in natural language. The generalized rules are expressed in terms of high level concepts for simplicity, conscience and clarity. Background knowledge, such as conceptual hierarchies etc., is generally available for knowledge discovery process.
Contents

1 Introduction 1

2 Primitives for Knowledge Discovery in Databases 3
   2.1 Data Relevant to the Discovery Process 3
   2.2 Background Knowledge 3
   2.3 Representation of Learning Results 5
   2.4 Language for Database Learning 6

3 The Concept Hierarchy 8
   3.1 Basics 8
   3.2 Role of Concept Hierarchy 8
   3.3 The Dynamic Hierarchy Adjustment Algorithm 8
   3.4 Drawbacks 10

4 Principles for Attribute-Oriented Induction 11
   4.1 Basic Strategies for Attribute-Oriented Induction 11
   4.2 Basic Attribute-Oriented Induction Algorithm 15
      4.2.1 Basic Strategies and Steps 15
      4.2.2 The Algorithm and Its Explanation 15

5 Learning Other Knowledge Rules by Attribute-Oriented Induction 17
   5.1 Learning Discrimination Rules 17
   5.2 Learning Data Evolution Regularities 18

6 Application of Discovered Rules 20

7 Dealing with Natural Language Queries 21
   7.1 Semantic Tractability 21
   7.2 An Overview of the Working 22

8 Architecture of the Interface System 25
   8.1 Lexicon 25
   8.2 Tokenizer 25
   8.3 Matcher 25
   8.4 Parser Plug In 26
   8.5 Query Generator 26
   8.6 Equivalence Checker 26

9 Theoretical Aspects 27

10 Implementation and Working 29
1 Introduction

Knowledge discovery from databases is the automated extraction of useful and interesting, implicit information from large bodies of diverse data stored in databases. The information to be retrieved could be sometimes previously unknown. 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. One primary method of extracting the information is by generalizing specific data values into more general concepts. The prime objectives of such discovery is to produce new non-trivial information that is understandable, accurate and useful. The information is implicit in the sense that it is contained in data, but hidden by a preponderance of overly specific data. 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.

Attribute oriented generalization seeks to transform diverse data stored in database relations into more general and useful information on an attribute by attribute basis. Generalization is accomplished by replacing the specific attribute data values of each tuple with more and more general concepts. In this way, the number of distinct values of each attribute is reduced as related values are grouped into more general concepts. Many tuples then become redundant as the information they contain becomes identical to the others tuples. These redundancies are eliminated by removing all but one of the identical tuples. This generalization continues until an acceptable level of information is attained.

The generalization is guided by concept hierarchies associated with each attribute of the relation to be generalized. A concept hierarchy is a tree structure defined by a domain except that provides a hierarchial taxonomy of concepts ranging from a single most general concept at the root of the tree to some representation of all possible attribute values at the leaves. Concepts are formed by grouping related attribute values together and representing all members of this set or range by a single symbol. These symbols in turn may be grouped into more general concepts which are again represented by another unique symbol. This grouping continues until the most general concept, called ANY, is reached.

The generalization process is limited by the specification of two tunable threshold values, the attribute threshold and the table threshold. The attribute threshold specifies the maximum number of distinct values of any attribute that may exist in the final generalized relation. Generalization proceeds first by reducing the number of distinct values of attributes to less than or equal to the attribute threshold. Duplicate tuples are then removed. The resulting relation is then called the prime relation. The table threshold specifies the maximum number of tuples that may exist in the final generalized relation. If the number of tuples in the prime relation exceeds the table threshold, then further generalization must take place. This is accomplished by choosing an attribute by some method, generalizing it another level, and then removing duplicates. This process iterates until the number of tuples is no more than the table threshold.

The result of the generalization, therefore, is a relation with a limited number of tuples containing much more general, summarized information than was in the original ungeneralized relation. Each tuple in this relation represents a class of the original tuples. Any of the tuples that were generalized into this final tuple are specific members of this class.
This generalization procedure is used to accomplish two types of learning tasks: characteristic learning tasks and classification or discriminate learning tasks. Characteristic learning tasks operate on one set of data and seek to discover interesting relationships between attributes of a relation. An example might be the relationship between the amount of money a researcher is granted and the area of study in which the researcher is working. Another example might be the relationship between observed symptoms and a disease. Classification or discriminate learning tasks operate on two sets of data and seek to determine interesting distinctions between the characteristics of the concepts represented in each. An example might be the amount of money granted to researcher in one field compared to the amounts of money granted to researchers in a different field. Another example might be symptoms that distinguish one disease from another.

The second aspect of this work is concentrated on the development of a natural language interface with the relational databases so as to facilitate laymen to use this system efficiently. The goal of using this natural language interface is to enable users put forward questions in natural language, precisely speaking in English. This English language questions will be translated into corresponding SQL queries that will perform the attribute-oriented induction on the concept hierarchy of the backend database to retrieve generalized relations, these generalized relations will, in turn, help the end-users to get back valuable data. While natural language sentences have the potential to be subtle, complex and ambiguous, they can also be simple, straightforward and clear enough to make interpretations. Basically we are dealing with the formalization of the intuition to answer by identifying classes of questions that are easy-to-understand in a well-defined sense.

To satisfy users, natural language interfaces can only misinterpret their questions very rarely if at all. This case is similar to that of a mouse that appropriately responds to a click most of the time, but periodically whisks the user to an apparently random location. Sometimes it might happen that if the system does not understand a user, it can indicate so and attempt to engage in a clarification dialog, but to actively misunderstand the user, to form an inappropriate SQL query, and provide the user with an incorrect answer, would erode the user’s trust and render the natural language interface unusable. So we aim to build a reliable interface that defines soundness and completeness and identifies a class of semantically tractable natural language questions. Taxonomic theory seeks to carve out classes of questions that are semantically tractable in the context of natural language processing. Natural language interfaces will not restrict their questions to a subset of English in practice, but rather, identifying classes of questions as semantically tractable (or not), and experimentally measuring the prevalence of such questions, is a worthwhile avenue of research for natural language interfaces.
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 discuss about relational databases in the following subsections.

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 the 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 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 data. 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 overgeneralization.

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 learnt, called the target class, from other classes, called the 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 regularities represent the characteristics of the changed data if it is a characteristic rule, or the features which discriminates 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 classes. The data in the contrasting classes imply that such data can not 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 organised 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 \textit{ANY}, and the most specific concepts correspond to the specific values of the attributes in the database. 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.

A concept hierarchy table of a student database of an Indian University looks like:

\{biology, chemistry, computing, ..., physics\} $\rightarrow$ science
\{literature, music, ..., painting\} $\rightarrow$ art
\{science, art\} $\rightarrow$ ANY(Subject)
\{freshman, sophomore, junior, senior\} $\rightarrow$ undergraduate
\{M.Sc., M.A., Ph.D.\} $\rightarrow$ graduate
\{undergraduate, graduate\} $\rightarrow$ ANY(Status)
\{Calicut, Cochin, Trivandrum, Palakkad\} $\rightarrow$ Kerala
\{Chennai, Coimbatore, Madurai, Tiruchirapally\} $\rightarrow$ Tamilnadu
\{Mumbai, Pune, Nagpur, Nasik\} $\rightarrow$ Maharashtra
\{Ahmedabad, Surat, Rajkot, Baroda\} $\rightarrow$ Gujarat
\{Melbourne, Sydney, Briabane, Canberra\} $\rightarrow$ Australia
\{Karachi, Lahore, Rawalpindi, Islamabad\} $\rightarrow$ Pakistan
\{Australia, England, Pakistan, ..., Sri Lanka\} $\rightarrow$ foreign
\{Kerala, Tamilnadu, Maharashtra, ..., Gujarat\} $\rightarrow$ India
\{foreign, India\} $\rightarrow$ ANY(place)
\{0.0 - 1.99\} $\rightarrow$ poor
\{2.0 - 2.99\} $\rightarrow$ average
\{3.0 - 3.49\} $\rightarrow$ good
\{3.5 - 4.0\} $\rightarrow$ excellent
\{poor, average, good, excellent\} $\rightarrow$ ANY(grade)

The concept hierarchy table of typical university database is given above where A $\rightarrow$ B indicates that B is a generalization of A. A concept tree represents a taxonomy of concepts of the values in an attribute domain. The concept tree for status of students can be summed in the following figure.

![Figure 1: A concept tree for Status](image-url)
Concept hierarchies can be provided by knowledge engineers of 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 “Trivandrum is a city of Kerala, which in turn is a province of India”, is usually stored in the database if there are attributes city, province and country. Such hierarchical relationships can be made explicit at the schema level by indicating “city — province — 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 hierarchial 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 he should give finer classification for the range 3 to 4. Even for attributes with discrete values, statistical techniques can be used under certain circumstances. For example, if the birth-place of most students are clustered in India and scattered in many other different countries, the highest level concepts of the attribute can be categorized into “India” and “foreign”. Similarly, an available concept hierarchy can be modified based on statistics. Moreover, the concept hierarchy of an attribute can also be discovered or refined based on its relationship with other attributes.

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 administrative regions such as provinces, countries, 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 processes.

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 learning results is called an intermediate generalized relation and which represents final learning results is called final generalized relation. In a generalized relation, some or all of its attribute values are generalized data, that is, non-leaf nodes in the concept hierarchies. Some learning-from-examples algorithms require the final learned rule to be in the conjunctive normal form. This requirement is usually unreasonable for large databases since the generalized data often contains 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 result may not be fully generalized. A small threshold value leads to a simple rule rule with few disjuncts. However, small threshold values may result in an overly generalized rule and some valuable information may be 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.

In a large relation, exceptional data often occur. it is important to consider exceptional cases when learning in databases. Statistical information helps learning-from-example to handle exceptions and *noisy data*. 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 an *quantitative rule*).

### 2.4 Language for Database Learning

Generalization can be performed in many different directions. Unconstrained learning may result in a very large set of learned rules. Moreover, different rules extracted from the same set of data using different background knowledge (concept hierarchy). 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 question in natural language for knowledge extraction by laymen,

[ii] a database query that extracts the relevant set of data,

[iii] the kind of rules to be learned,

[iv] the specification of the target class, and possible the contrasting classes depending on the rules to be learned,

[v] the concept hierarchy,

[vi] the preferred form to express the learning results.

If our aim is to learn a *discrimination rule* which distinguishes Ph.D students from M.Sc students in science based upon the level of courses in science which they appear for. The learning involves both the relations *Student* and *Course*. The request is specified in the following way:-
in relation Student S, Course C
learn discrimination rule for $S.status = \text{"Ph.D"}$
in contrast to $S.status = \text{"M.Sc"}$
where $S.subject = \text{"science"}$ and $C.department = \text{"science"}$ and $C.TA = S.name$
in relevance to $C.level$

A database query is embedded in the learning request, and $\text{science}$ is a piece of generalized data which can be found in the concept hierarchy table. The preferred conceptual hierarchy can be specified independently. Since neither of them is specified explicitly in this learning request, default hierarchies and thresholds are used.
3 The Concept Hierarchy

3.1 Basics
A concept hierarchy defines a sequence of mappings from a set of lower-level concepts to their higher level correspondence. Such mappings may organize the set of concepts in partial order, such as in the shape of a tree (a hierarchy, a taxonomy), a lattice, a DAG etc. although, they are still called *hierarchies* for convenience. Since, concept hierarchies define mapping rules between different levels of concepts, they are in general data or application-specific. Many concept hierarchies, such as birthplace (city, province, country) are actually stored implicitly in the database, such as in different attributes or different relations, which can be made explicit by specifying certain attribute mapping rules. For example, the floor area of a house can be computed from the dimension of each segment in the house by a spatial computation algorithm, and then mapped to a high level concept, such as, small, large etc. defined by deduction rules. Different concept hierarchies can be constructed on the same attribute(s) based on different viewpoints or preferences. For example, the birthplace could be organized according to geographical locations and administrative regions, size of cities etc.

3.2 Role of Concept Hierarchy
The concept hierarchy plays the following important roles:-

[i] Retrieval of the relevant set of data.
[ii] Determination of generalization pairs in the derivation of the prime relation.
[iii] Further generalization of prime relations.

3.3 The Dynamic Hierarchy Adjustment Algorithm
A concept hierarchy consists of a set of nodes organized in a partial order. A node is a *leaf node* if it has no children, or a *non-leaf node* otherwise. An *occurrence count* of a node is a number of associated nodes with that particular node. It represents, if it is a *leaf node*, the number of occurrences of the value in the task-relevant data set, or if a *non-leaf node*, then the sum of the *occurrence counts* of its children nodes. A total occurrence of an attribute is the sum of the *occurrence counts* of all the leaf nodes in the initial data relation.

**Input:**  
[i] A learning task-relevant initial relation \( W \),  
[ii] an attribute \( A \),  
[iii] the *attribute threshold* \( T \) (maximum number of allowable disjuncts) for attributes,  
[iv] a prespecified concept hierarchy \( H \).

**Output:** An adjusted concept hierarchy \( H^* \) of attribute \( A \) for the derivation of the prime relation and for further generalization.

**Method:** The adjustment algorithm essentially consists of two processes:-  
[i] top-down *big* nodes promotion, and  
[ii] bottom-up *small* nodes merging.

**Steps of phase I:**

[1] **Initialization:**

(a) Assign level numbers to each node in the hierarchy \( H \) according to the given partial order.
(b) Scan once the corresponding attribute of each tuple in the initial relation W, calculate the occurrence count C, count for each leaf node and propagate them to the corresponding parents in the hierarchy H. The total occurrence is the sum of the counts of all the leaf nodes in the hierarchy. Notice only non-zero count nodes are considered in the above following computations.

[2] Top-down traversal of concept hierarchy, H::
(a) Set a buffer set, prime, initially empty, and another buffer set, buf, to hold the set of nodes at the top level of H:-
(i) Calculate the weight and count of each node.
(ii) Set weight threshold equal to the attribute threshold.
(iii) Perform node marking. A node, if weighted not less than T, is a big node, otherwise a small one. Do separate markings for big leaf node-s, big non-leaf node-s, small leaf node-s and small non-leaf node-s.
   (b) Call expand buffer, which is implemented as follows:-
      (i) Move every big leaf node from buf to prime.
      (ii) Replace every big leaf node by its children.
      (iii) Repeat this process until no more changes (i.e. only small leaf node-s or small non-leaf node-s are left in buf).
   (c) Perform weight recalculation and node re-marking again as follows:-
      (i) If the weight considering prime and buf together is less than the weight threshold value, then move all the nodes from buf to prime and the process terminates. Otherwise, set the weight threshold value to itself plus that of prime, total to the sum of the counts in buf and weight of each node in buf to count.
      (ii) Re-mark the nodes based on their weights and repeat expand buffer and weight recalculation process until no more changes.

[3] Bottom-up traversal of concept hierarchy, H::
(a) If there are still nodes left in buf, perform a bottom-up merging of the remaining nodes in buf as follows:-
   (i) Starting at the bottom level, step up one level (suppose, to level i) and merge the nodes in buf which share a common ancestor at level i.
   (ii) If the weight of the merged node is no less than the weight threshold, move it to prime and decrement the weight threshold.
   (iii) If the total number of nodes in buf is no more than the weight threshold, then move all the nodes in buf to prime. Else perform weight recalculation, step up a level and repeat the process.
   (iv) If there is no more level to climb (i.e. the hierarchy is in the shape of a forest), group the nodes into T groups and move them to prime.
   (b) Follow the different available conventions while naming a merged node:-
      (i) A node is named A+B, if it is the result of merging two nodes A and B.
      (ii) A node is named E-A, if it is equivalent to an existing node E with one child removed.
      (iii) A node is named E* if it is equivalent to an existing node E with more than child nodes removed.

[4] Linkage::
Build up the generalization linkage between the nodes in prime and the attribute data in
the initial relation.

**Steps of phase II:**

[1] There are more than $T$ (attribute threshold) nodes in $prime$, and there exists a generalization linkage between every node in the initial relation and a node in $prime$ after the execution of the algorithm.

[2] According to the algorithm, every node moved into $prime$ must satisfy one of the following three conditions:
   a) A node with weight greater than 0.
   b) The combined weights of $prime$ and $buff$ is no more than $T$.
   c) The remaining nodes are grouped into $T$ groups (i.e. $T$ new nodes) when there is no more level to climb.

Moreover, the computation of $T$ ensures that the number of the accumulated nodes is no more than $T$.

### 3.4 Drawbacks

[1] It can not generate more than $T$ nodes in $prime$.

[2] Every non-zero count node is either a leaf node moved into $prime$, or is associated with a non-leaf node (ancestor) that is finally moved into $prime$.

[3] There should exist a generalization linkage from the node to a node in the prime relation after the execution of the algorithm.

[4] Nodes in the $prime$ buffer should carry relatively even data distribution, and the shape of the hierarchy should be maximally preserved.
4 Principles for Attribute-Oriented Induction

4.1 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 as follows.

<table>
<thead>
<tr>
<th>name</th>
<th>status</th>
<th>subject</th>
<th>birth−place</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>rahul</td>
<td>M.A</td>
<td>history</td>
<td>trivandrum</td>
<td>3.5</td>
</tr>
<tr>
<td>anand</td>
<td>senior</td>
<td>math</td>
<td>cochin</td>
<td>3.7</td>
</tr>
<tr>
<td>abdul</td>
<td>junior</td>
<td>liberal arts</td>
<td>lahore</td>
<td>3.6</td>
</tr>
<tr>
<td>vinod</td>
<td>junior</td>
<td>physics</td>
<td>ahmedabad</td>
<td>3.9</td>
</tr>
<tr>
<td>steve</td>
<td>M.Sc</td>
<td>math</td>
<td>melbourne</td>
<td>3.3</td>
</tr>
<tr>
<td>mark</td>
<td>Ph.D</td>
<td>chemistry</td>
<td>canberra</td>
<td>2.7</td>
</tr>
<tr>
<td>tony</td>
<td>sophomore</td>
<td>chemistry</td>
<td>mumbai</td>
<td>3.5</td>
</tr>
<tr>
<td>monica</td>
<td>Ph.D</td>
<td>computing</td>
<td>karachi</td>
<td>3.4</td>
</tr>
<tr>
<td>ajay</td>
<td>M.Sc</td>
<td>biology</td>
<td>colombo</td>
<td>3.0</td>
</tr>
<tr>
<td>sachin</td>
<td>Ph.D</td>
<td>music</td>
<td>chennai</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>....</td>
<td>..........</td>
<td>............</td>
<td>....</td>
</tr>
<tr>
<td>michael</td>
<td>freshman</td>
<td>statistics</td>
<td>sydney</td>
<td>3.9</td>
</tr>
</tbody>
</table>

Figure 2: 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, Subject, Birth-Place and GPA, using the default conceptual hierarchy in Fig. 1 and the default threshold value of 3. Suppose the learning task is represented by:-

\[
\begin{align*}
\text{in relation student} \\
\text{learn characteristic rule for status = “graduate”} \\
\text{in relevance to name, major, birth-place, GPA}
\end{align*}
\]

To learn the characteristic rule for the Status = “graduate”, preprocessing is performed by first selecting graduate students. Since graduate is a non-leaf node in the concept hierarchy on
<table>
<thead>
<tr>
<th>name</th>
<th>subject</th>
<th>birth-place</th>
<th>GPA</th>
<th>vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>rahul</td>
<td>history</td>
<td>trivandrum</td>
<td>3.5</td>
<td>1</td>
</tr>
<tr>
<td>steve</td>
<td>math</td>
<td>melbourne</td>
<td>3.3</td>
<td>1</td>
</tr>
<tr>
<td>mark</td>
<td>chemistry</td>
<td>canberra</td>
<td>2.7</td>
<td>1</td>
</tr>
<tr>
<td>monica</td>
<td>computing</td>
<td>karachi</td>
<td>3.4</td>
<td>1</td>
</tr>
<tr>
<td>ajay</td>
<td>biology</td>
<td>colombo</td>
<td>3.0</td>
<td>1</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>............</td>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>sachin</td>
<td>music</td>
<td>chennai</td>
<td>3.8</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3: The initial data relation for induction

Status, the hierarchy table should be consulted to extract the set of the corresponding primitive data stored in the relation, which is M.Sc, M.A, Ph.D. Then the data about graduates can be retrieved and projected on relevant attributes Name, Subject, Birth-Place and GPA, which results in an initial data relation on which induction can be performed. The next table 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. The 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.

Generalization is a process of learning from positive examples. Generalization on the smallest decomposable components rather than on composite attribute ensures that the smallest possible changes 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 the general properties of a graduate student can not be characterized by the attribute name.

**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.

This strategy corresponds to the generalization rule, dropping conditions, in learning-from-examples. 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
can not be generalized using higher level concepts, thus the attributes should be removed. The
three remaining attributes subject, birth-place and GPA can be generalized by substituting for
subordinate concepts by their corresponding superordinate concepts. For example, physics can
be substituted by science and chennai by tamilnadu.

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

This strategy corresponds to the generalization rule, *climbing generalization trees*, in learning-
from-examples. The substitution of an attribute value by its higher level concept makes the
tupule cover more cases than the original one and thus generalizes the tupule. 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 tupules may generalize to an identical tupule where two tupules 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
when merging identical tupules.

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

By definition of vote, the vote of each generalized tupule must register the number of the
tupules in the initial data relation generalized to the correct one. Therefore, to keep the correct
votes registered, the vote of each tupule should be carried in the generalization process, and
such votes will be accumulated when merging identical tupules.

<table>
<thead>
<tr>
<th>subject</th>
<th>birth-place</th>
<th>GPA</th>
<th>vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>kerala</td>
<td>excellent</td>
<td>35</td>
</tr>
<tr>
<td>science</td>
<td>tamilnadu</td>
<td>excellent</td>
<td>10</td>
</tr>
<tr>
<td>science</td>
<td>kerala</td>
<td>excellent</td>
<td>30</td>
</tr>
<tr>
<td>science</td>
<td>Australia</td>
<td>good</td>
<td>15</td>
</tr>
<tr>
<td>science</td>
<td>Pakistan</td>
<td>good</td>
<td>10</td>
</tr>
</tbody>
</table>

*Figure 4: A Generalized Relation*

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

**Strategy 6: (Threshold control on generalized relation)** 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.

Further generalization should be performed if the number of tuples in a generalized relation is greater than the threshold value. By further generalization on selected attributes 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. The generalized relations can be examined by users or experts interactively to filter out trivial rules and preserve interesting ones. In the previous table, 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, resulting in the following relation:-

<table>
<thead>
<tr>
<th>subject</th>
<th>birth-place</th>
<th>GPA</th>
<th>vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>India</td>
<td>excellent</td>
<td>35</td>
</tr>
<tr>
<td>science</td>
<td>India</td>
<td>excellent</td>
<td>40</td>
</tr>
<tr>
<td>science</td>
<td>foreign</td>
<td>good</td>
<td>25</td>
</tr>
</tbody>
</table>

Figure 5: Further Generalization of the relation

**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.

Simplification can be performed by unioning the first two tuples if set representation of an attribute is allowed, as in the following figure.

Suppose *art* and *science* cover all of the subject areas. Then \{art, science\} can be generalized to ANY and be removed from the representation. Therefore the final generalized relation is equivalent to rule that, a graduate is either (with 0.75 probability) an Indian with an excellent GPA or (with 0.25 probability) a foreign student of science with a good GPA.
4.2 Basic Attribute-Oriented Induction Algorithm

4.2.1 Basic Strategies and Steps

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

Input: [i] A relational database, [ii] The learning task, [iii] The preferred concept hierarchies, and [iv] The 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.
Step 4. Transform the final relation into a logical rule.

4.2.2 The Algorithm and Its Explanation

begin { basic attribute-oriented induction }
  for each attribute $A_i$ in the generalized relation GR (where i varies from 1 to n, n = number-of-attributes ) do {
    while no.-of-distinct-values-in-$A_i$ is greater than 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 tupules }
    while no.-of-tupules in GR is greater than threshold do {
      selectively generalize attributes; merge identical tupules }
  }
end

This algorithm learns correctly characteristic rules from relational databases. In step 1, the relevant set of data 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, step 3 and step 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.
5 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.

5.1 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. It is illustrated in the following example.

Example: Suppose a discrimination rule is to be extracted to distinguish graduate students from undergraduate students in the relation Student discussed before. Clearly, both the target class graduate and the contrasting class undergraduate are relevant to this 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.

As per the above table, 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 undergraduate are overlapping tuples, which indicates that a Kerala-born student, with subject science and 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.

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.

Since the overlapping tuples represent the same assertions in both the target class and the contrasting class, the concept described by the overlapping tuples can not 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 discrimination 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 undergraduate, the target class contains four unmarked unmarked tuples as in the previous figure, 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 follows.

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
<table>
<thead>
<tr>
<th>class</th>
<th>subject</th>
<th>birth-place</th>
<th>GPA</th>
<th>vote</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>graduate</td>
<td>art</td>
<td>kerala</td>
<td>excellent</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>tamilnadu</td>
<td>excellent</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>kerala</td>
<td>excellent</td>
<td>30</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>australia</td>
<td>good</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>pakistan</td>
<td>good</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>undergraduate</td>
<td>science</td>
<td>gujarat</td>
<td>excellent</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>art</td>
<td>gujarat</td>
<td>average</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>kerala</td>
<td>average</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>kerala</td>
<td>excellent</td>
<td>35</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>art</td>
<td>kerala</td>
<td>average</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>art</td>
<td>tamilnadu</td>
<td>excellent</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: A generalized relation

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.

5.2 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, $DB_{t1}$ and $DB_{t2}$, where $t1$ and $t2$ are not equal, are required for such knowledge discovery. Our discussion can be generalized to multiple 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 characteristics of the relevant data in the current database from those in a previous database instance. Attribute-oriented induction can be used in the generalization process.

**Example:** 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: collecting task-relevant
data and 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 non-primitive concept, data retrieval should be performed by consulting the concept hierarchy as well. The second phase is carried out in the same manner as the previously studied attribute-oriented induction for learning characteristics rules.

Suppose that another learning request is to distinguish the characteristics of the undergraduate students enrolled in July 1998 from those enrolled in July 2002. The knowledge discovery process can still be partitioned into two phases: collecting task-relevant data and grouping them into two classes, the target class and the contrasting class and then, performing attribute-oriented induction synchronously on the two classes. The first phase is performed by finding all of the undergraduate students enrolled in July 1998 and those enrolled in July 2002 and grouping them into two classes respectively. The second phase is the same as the previous one.

Such 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.

<table>
<thead>
<tr>
<th>class</th>
<th>subject</th>
<th>birth-place</th>
<th>GPA</th>
<th>vote</th>
<th>mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>graduate</td>
<td>art</td>
<td>india</td>
<td>excellent</td>
<td>35</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>india</td>
<td>excellent</td>
<td>40</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>foreign</td>
<td>good</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>india</td>
<td>excellent</td>
<td>50</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>art</td>
<td>india</td>
<td>average</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>science</td>
<td>india</td>
<td>average</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>art</td>
<td>india</td>
<td>excellent</td>
<td>20</td>
<td>*</td>
</tr>
</tbody>
</table>

Figure 8: Further generalized relation
6 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.

Database knowledge represents the semantic information associated with databases, which includes deduction rules, integrity constraints, concept hierarchies about data and general data characteristics. Cooperative query answering consists of analyzing the intent of query and providing generalized, neighbourhood or associated information relevant to the query. Semantic query optimization applies database semantics, integrity constraints and knowledge rules to optimize queries for efficient processing. Previous studies on querying database knowledge and intelligent query answering were focussed 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.

Suppose we are asked to describe the characteristics of the graduate students in computing science who were born in India with excellent academic performance. It is to be noticed that graduate, India 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.Sc, M.A, Ph.D \}. 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 instead on the primitive data in the database.

Moreover, semantic query optimization can be performed on queries using database semantics, concept hierarchies and the discovered knowledge rules. Let the knowledge rules discovered in our previous examples be stored in the database, and the new query is to find all the foreign students born outside India with subject science an GPA between 3.2 to 3.4. We have already discovered the knowledge that all of the foreign students with subject science and a good GPA must be graduate students. Since this knowledge 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.
7 Dealing with Natural Language Queries

7.1 Semantic Tractability

A database is built up of three types of elements: relations, attributes and values. Each element is distinct and unique. An attribute element is a particular column in a particular relation and each value element is the value of a particular attribute. A value is compatible with its attribute and also with the relation containing the attribute. An attribute is compatible with its relation. Let a wh-word be any word in the set {“what”, “which”, “where”, “who”, “when”}. Each database attribute has a set of compatible wh-values.

A token is a set of word stems that matches a database element. For instance, {employee, salary} and {employee, remuneration} match the database attribute employee salary. Many different tokens might match the same database element, and conversely, a token might match match several different elements (sometimes with different types). Each token has a set of possible types (e.g, value tokens, attribute tokens) corresponding to the database elements it matches. A syntactic marker (such as “the”) is a token that belongs to a fixed set of database-independent tokens that make no semantic contribution to the interpretation of a question.

Now, we begin to formalize the notion of semantically tractable questions. Our definition is based on the observations that many natural questions specify a set of attribute/value pairs as well as free-standing values, where the attribute is implicit. An attribute (or relation) may also be paired with a wh-word. For example, suppose the natural language question is “what five-star hotels are located in Ooty?” in the context of of a database containing the relation Hotels with attributes name, rating, location. The word “five-star” refers to the value “five-star” of the implicit database attribute rating, similarly, the words “located” and “Ooty” refer to the explicit attribute location and its value “Ooty”, and the word “hotel” refers to the relation Hotels and corresponds with the wh-word “what”. The fact that attributes may be implicit allows a form of “ellipsis” that occurs frequently in questions; recovering this missing information is an important challenge for semantic interpretation.

The definition of a semantically tractable question captures the above intuitions. First, we require that some set of tokens exists such that every word in the question appears in exactly one token. We refer to any such token set as a complete tokenization of the question. We can use an attachment function to model syntactic attachment constraints derived from the question’s parse tree. Attachment is a function formpairsof tokens to TRUE or FALSE. For instance, in the above example the tokens “located” and “Ooty” are attached while the tokens “what” and “Ooty” are not.

In order for the sentence to be interpreted in the context of the given database, at least one complete tokenization must map to some set of database elements E as follows:-

[1] Each token matches a unique database element in E. This means that there is a one-to-one match between the tokens in the tokenization and E.

[2] Each attribute token corresponds to a unique value token. This means that:- (a) the database attribute matching the attribute token and the database value matching the value token are compatible, and (b) the attribute token and the value token are attached. For instance, consider the tokens “locate” (which matches the attribute location) and “Ooty” (which matches the value “Ooty”). “Ooty” is compatible with location and the two given tokens are attached. Thus,
“locate” \textit{corresponds} to “Ooty”.

[3] Each relation token \textit{corresponds} to either an attribute token or a value token. This means that: (a) the database relation matching the relation token and the database element matching the attribute or value token are compatible, and (b) the relation token is attached to the corresponding attribute or value token. For instance, the relation token “hotel” (which matches the relation \textit{Hotels}) corresponds to the attached value token “five-star” (which matches the value “Five-star” corresponding to the attribute \textit{rating}).

A value token need not have a corresponding attribute token in the sentence, which is the form of the “ellipsis” method mentioned earlier. A mapping from a complete sentence tokenization to a set of database elements such that conditions 1 through 3 are satisfied is a \textit{valid mapping}. If the sentence tokenization contains only distinct tokens and at least one of its value tokens matches a \textit{wh}-value, we refer to the corresponding sentence as \textit{semantically tractable}.

### 7.2 An Overview of the Working

Given a question \(q\) in natural language, we have to determine whether it is semantically tractable and if so, to produce the output corresponding SQL query (or, queries). The problem of finding a mapping from a complete tokenization of \(q\) to a set of database elements such that the semantic constraints imposed by conditions 1 through 3 are satisfied is reduced to a graph matching problem. We can use the max-flow algorithm to efficiently solve this problem. Each max-flow solution corresponds to a possible semantic interpretation of the sentence. We have to collect the max-flow solutions, discard the solutions that do not obey syntactic constraints, and retain the rest as the basis for generating SQL queries corresponding to the question \(q\).

Let us consider the mapping of the example question, “What are the security-settings of AMD on a WINDOWS system?” to an SQL query. There is ambiguity in this question which we have to solve automatically. For brevity and clarity, this example refers to a single relation \textit{Security-settings} with attributes \textit{Description}, \textit{Platform} and \textit{Company}. After the discussion, a slightly modified version of this sentence is also discussed to handle multiple relations.

The tokenizer produces a single complete tokenization of the question: (“what”, “AMD”, “security-settings”, “WINDOWS”, “system”). It is to note that the tokenizer strips syntactic markers such as “the” and “a”. By looking up the tokens in the lexicon, the set of matching database elements for every token is retrieved. In this case, “what”, “AMD” and “WINDOWS” are value tokens, “system” is an attribute token and “Security-settings” is a relation token.

Next, the matcher constructs the attribute-value graph shown in the next figure. To understand the meaning of nodes in the graph, it is helpful to read it column by column from left to right. The leftmost node is a source node. The Value Tokens column consists of the tokens matching database values (which in turn can be found in the DB Values column). For instance, the token AMD is ambiguous as it could either match a value of the \textit{Company} attribute or a value of the \textit{Platform} attribute. Edges are added from each value token to each matching database value. Solid edges represent the final flow path while the dashed edges suggest alternative flow routes. Let \(F\) denote the flow in the network.

The matcher connects each database value to its corresponding database attribute. Each attribute is then connected to its matching attribute tokens and also to the node I, which
What are the security-settings of AMD on a WINDOWS system?

### LEXICON
- security
- settings
- description
- platform
- AMD
- WINDOWS
- company
- developer
- administrator
- what

### SELECT DISTINCT Description FROM Security-settings WHERE Company = ‘AMD’
AND Platform = ‘WINDOWS’

Figure 9: The transformation of the given natural language question to an SQL query

stands for implicit attributes. All attribute tokens link to the node E, which stands for explicit attributes. Finally, both E and I link to the sink node T.

Notice that two instance of the column containing DB attribute nodes. The unit edge from each DB attribute node to itself ensures that only one unit of flow in fact traverse each such node. These edges are needed because more than one DB value is compatible with a given DB attribute and a DB attribute may match more than one attribute token, however, our definition of a valid mapping requires each DB attribute be used only once.

The graph is interpreted as a flow network where the capacity on each edge is 1, unless otherwise indicated. the capacity on the edge from E to T is the number of attribute tokens (here, 1). The capacity on the edge from I to T is the number of value tokens minus the number of attribute tokens. That difference is 2 in our example. Setting the capacity to be this difference forces the max-flow algorithm on the graph subject to these capacity constraints and searching for an integer solution. The maximum flow through the network in this network is 3. In fact, the maximum flow in any graph constructed in this way is equal to the number of value tokens because each value token has to take part in the match produced by this algorithm.

The solid arrows indicate the path chosen by the maxflow algorithm. It is notable how the ambiguity regarding whether AMD is the name of a company or of a platform is resolved by maximizing the flow. The algorithm “decides” that AMD is the name of a company because this choice allows flow along two edges with capacity 1 into node I. Because the edge (I,T) has
capacity 2, this choice maximizes the flow through the graph (F=3). If the algorithm “decided” that AMD was the platform, there would be no possible interpretation for “WINDOWS” and the final flow would be 2.

After all attribute and value tokens have been matched to database elements, we have to ensure that all relation tokens correspond to either a value token or an attribute token. In the case of a unique relation token (Security-settings), this amounts to checking whether any of the matching database relations contains some attribute matching an attribute token. Since in our example, “security-settings” matches only “Security-settings”, the algorithm has found a one-to-one match between the sentence tokens and the database elements that satisfies the semantic constraints in the set of conditions for semantically tractable sentences. at this point, we have to check whether the match satisfies the syntactic constraints represented by pairs of attached tokens: (“what”, “security-settings”), (“AMD”, “security-settings”) and (“WINDOWS”, “system”). IF all the attachment constraints are satisfied, it means that a valid mapping has been found. Each valid mapping is converted into a SQL query.

Consider slightly modified version of the above sentence: “What are the security-settings of AMD on a WINDOWS system for a virus attack?” in the context of a database containing
a Security-settings as well as a Threat table. In addition to the Description, Platform and Company attributes, Security-settings also contains a Type field. Threat has attributes Name, Agents and Type, which is a foreign key corresponding to Security-settings.Type.

Using the lexicon we get at first that “attack” is a synonym for “threat”. Then we have to build an attribute-valued graph similar to the previous one. This time, the first node column contains a node for the value token “virus”, which matches the value “virus” of the database attribute Threat.Agent. Since we now have 4 value tokens and 1 attribute token, the capacity of the edge (E,T) remains 1 while the capacity of (I,T) is 3. A unique maximum-flow solution with F=4 exists and suggests the following token-database element mapping: “AMD” matches Security-settings.Company = AMD, “WINDOWS” matches Security-settings.Platform = WINDOWS, “virus” matches Threat.Agent = virus and “system” matches Security-settings.Platform.

Since all attribute and value tokens have been matched to database elements, now we must ensure that all relation tokens correspond to either a value token or an attribute token. We solve the problem by building an additional relation flow network. We record the database relations employed by the attribute/value max-flow solution: Security-settings and Threat. A relation flow network with unit-capacity edges is constructed as in the previous figure. The max-flow solution in this network is found such that \( F = \text{number of relation tokens} = 2 \). Such a solution yields a one-to-one match of relation tokens to database relations that contain attributes and values that match sentence attribute tokens and value tokens. Moreover, we need to ensure that a valid mapping has been found.

No other valid mappings are found and the query generator constructs the SQL interpretation(s) corresponding to the given sentence. When the valid mapping contains multiple database relations, a different SQL interpretation is built for each join path corresponding to the set of relations. In our example, the assumption is that the database contains only two tables Security-settings and Threat and the only possible join path is Security-settings.Type = Threat.Type. Hence the following SQL query is built:-

Figure 11: The Relation graph for the modified question
```
select distinct Security-settings.Description
from Security-settings, Threat
where (Security-settings.Company = ‘AMD’) and (Security-settings.Platform = ‘WINDOWS’)
and (Threat.Agent = ‘virus’) and (Security-settings.Type = Threat.Type);
```
8 Architecture of the Interface System

Let us have a look at the various components of the system:-

![System architecture diagram]

Figure 12: System architecture

8.1 Lexicon

The lexicon supports the following two operations:-
[1] given a word stem ws, retrieve the set of tokens which contain ws.
[2] given a token t, retrieve the set of database elements matching t.

In the following, we describe the manner in which the lexicon is derived from the database. The name of all database elements are extracted and split into individual words. Each word is then stemmed and a corresponding set of synonyms is identified using a general-purpose word ontology (Word Net). Each database element is thus associated with a set of word stems and each word stem is in turn associated with a set of synonyms. For every set $s_d$ of stemmed words corresponding to a database element d, a \textit{synonym-augmented} set of word stems $\text{syn}_d$ is computed by successively replacing each word stem in $s_d$ with a possible synonym. For instance, the set \{“require”, “experience”\} will yield the \textit{synonym-augmented set} \{“need”, “experience”\}. Any \textit{synonym-augmented set} of words $\text{syn}_d$ represents a token matching the database element d. In the above example, \{“require”, “experience”\} and \{“need”, “experience”\} are tokens...
matching the database attribute Required Experience. All the tokens derived in this manner point to hash tables containing the database elements they match. Furthermore, the tokens are placed in a hash table indexed by word-stems - this two-level storage structure ensures that the lexicon can easily perform its two defining operations.

8.2 Tokenizer

The tokenizer’s input is a natural language question and its output is the set of all possible complete tokenizations of the question. The tokenizer proceeds by stemming each word in the question, and the looking up in the lexicon the set of tokens containing the word stem. For each potential token, the tokenizer checks whether the other words in the token are also present in the question. For example, the word “price” is contained in tokens matching several database attributes \{Breakfast.Price, Lunch.Price, Dinner.Price\}. However, when the question contains the phrase “price of breakfast”, the only relevant attribute is Breakfast.Price and the only relevant token is “price breakfast”. Finally, the tokenizer also assigns to each token the types of database elements it could potentially match to (e.g., value, attribute, relation).

Once potential tokens are identified, computing the set of complete tokenization is equivalent to the NP-hard problem of exact set covering. In practice, however, the average number of complete tokenizations is close to 1 and tokenization takes less than 2 seconds (wall-clock time).

8.3 Matcher

The matcher embodies the key innovation in this architecture. We reduce the problem of finding a semantic interpretation of ambiguous natural language tokens as database elements to a graph-matching problem. More precisely, our reduction is to a maximum-bipartite-matching problem with the side constraints that all value token and attribute token nodes and a specific subset of the Db value and DB attribute nodes be involved in the match. The matcher runs in polynomial time in length of the natural language question and in the maximum ambiguity of question tokens.

8.4 Parser Plug In

We have to extract attachment relationships between tokens from the parse-tree. For example, the parser enables the system to realize that in the question, “What are the capitals of the Indian states?”, the token “capital” is attached to the token “state”. The attachment relationships are used by the matcher in the generation of valid mappings (only semantic interpretations which satisfy the syntactic attachment constraints represent valid mappings).

8.5 Query Generator

The query generator takes the database elements selected by the matcher and weaves them into a well-formed SQL query. In the case of single-relation queries, this process is straightforward. The SELECT portion of the query contains the database elements paired with wh-words; the
WHERE portion contains a conjunction of attributes and their values and the FROM portion contains the relevant relation name for the attributes in WHERE. In the case of multiplrelation queries, the generator adds the condition of join to the WHERE clause, which reflect a join path that contains all the relations implicitly invoked by attributes in the query. The participating relation names are also listed in the FROM clause. If the join path is unique, the generator terminates. Otherwise, the generator generates a query for each possible join path and submits the queries to the equivalence checker. It is possible for multiple join paths to yield SQL queries that are not equivalent, hence ambiguity arises.

8.6 Equivalence Checker

The equivalence checker tests whether there are multiple distinct solutions to the maxflow problem and whether these solutions translate into distinct SQL queries. Finding distinct maxflow solutions amounts to finding all maximum-matchings in two bipartite subgraphs of the current flow-graph. We employ a well-known maximum-matching enumeration algorithm. We check the query equivalence using this algorithm, which is polynomial time for acyclic conjunctive queries. If there are two distinct SQL queries, it is not possible to output an answer as the system cannot be certain which query is the right one. Hence one particular feature can be accommodated to ask the user to choose the correct interpretation of the question put forward by him.
9 Theoretical Aspects

In this section we will define the class of semantically tractable questions. Given a set of database elements $E$, let $E_v$, $E_a$ and $E_r$ denote the sets of values, attributes and relations in $E$. Given a set of tokens $T$ and a set of database elements $E$ such that each token in $T$ refers to a unique matching element in $E$, let $T_v$, $T_a$ and $T_r$ denote the sets of value tokens, attribute tokens and relation tokens in $T$.

We can now define the notion of a valid mapping:
Given a question $q$ with a tokenization $T$, attachment function $AT$, a lexicon $L$, and a set of database elements $E$, we say that there is a valid mapping form $T$ to $E$ if the following conditions hold:

1. **Sentence Token - Database Element Match**
   There is a one-to-one match (respecting $L$) between the tokens in $T$ and the database elements in $E$.

2. **Attribute Token - Value Token Correspondence**
   Each attribute token corresponds to a unique attached value token. Formally, all $T_a$ tokens can be assigned to value/attribute token pairs $(t_v, t_a)$ (no token appearing more than once) such that $t_v, t_a$ are attached and for each $(t_v, t_a)$ pair, a unique value/attribute pair $(d_v, d_a)$ exists in $E_{av}$ such that $t_v$ and $t_a$ match respectively $d_v$ and $d_a$.

3. **Implicit Attributes**
   Each value token matches to a database value that is compatible with some database attribute. Some of these database attributes may not match any of the attribute tokens in the sentence; we refer to them as implicit attributes. Formally, $E$ can be extended to set $E^*$ by adding attribute elements such that $E^*$ can be grouped into: a set $E_r$ consisting of distinct database relations, a set $E_{av}$ of compatible attribute/value pairs, with no database element appearing more than once such that each relation in $E_r$ contains at least one attribute in $E_{av}$.

4. **Relation Token - Attribute/Value Token Correspondence**
   Each relation token corresponds to either an attached attribute token or an attached value token. Formally, each $T_r$ token can be assigned to either an attribute token or a value token, creating the pair $(t_r, t)$ with the following properties: (a) the elements of the pair are attached, (b) $t_r$ matches a relation element $d_r$ in $E_r$ and $t$ matches an element $d$ of some $E_{av}$ pair such that $d$ and $d_r$ are compatible.

Given a valid mapping for a question $q$, it is a straight forward syntactic manipulation to construct the SQL interpretation.

Finally, we can introduce our key taxonomic distinction. A question $q$ is said to be semantically tractable relative to a given lexicon $L$, and an attachment function $AT$ if and only if $q$ has at least one complete tokenization $T$ such that:

- All tokens in $T$ are distinct.
- $T$ contains at least one wh-token.
- There exists a valid mapping (respecting $AT$ and $L$) from $T$ to some set of database elements.
E.
A semantic interpretation algorithm is said to be *sound* if it returns only valid mappings; the algorithm is said to be *complete* if it returns all valid mappings. The properties of soundness and completeness are essential to natural language interface reliability issue. Soundness blocks erroneous interpretations of the user’s questions, and completeness ensures that when the user’s question is ambiguous (i.e. it has several distinct interpretations) the system can detect the ambiguity and respond appropriately.
10 Implementation and Working

For implementing this project, the above design principle should be followed out and out. The underlying language in which the codes are written is java. We will be using the WINDOWS 2000 Professional platform. And for database operations we will be using SQL Plus of Oracle 8i version. The entire work can be divided into three main stages:-

[1] Processing the question in natural language.

At first, the natural language query is transformed into corresponding SQL query. For extraction of synonyms, the dictionary interface WordNet is used. The connection with the backend database is done using the driver Microsoft ODBC for Oracle. A few sample backend databases for verification are prepared and pasted to Oracle and the tables are formed in Oracle. Now, we have in our hand, a database containing relations and the relevant task what needs to be done. The backend knowledge and as well, the task-relevant data are stored there in the database. Now, to retrieve knowledge, we have to generalize our task. For that, we move to the next stage.

The next stage deals with building the hierarchy of the data relations. Using the Dynamic hierarchy adjustment algorithm, the job is accomplished. All the buffers needed and nodes are implemented. As a result, we get a generalized relation for the prime attributes. On these generalized relations, we have to apply the attribute-oriented induction algorithm to retrieve the knowledge.
References


