Knowledge Discovery in Object-Oriented Databases: The First Step *

Shojiro Nishio† Hiroyuki Kawano‡ Jiawei Han§

† Department of Information Systems Engineering, Osaka University, Osaka 565, Japan (nishio@ise.osaka-u.ac.jp)

‡ Department of Applied Math. and Physics, Kyoto University, Kyoto 606, Japan (kawano@kuamp.kyoto-u.ac.jp)

§ School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6 (han@cs.sfu.ca)

Abstract

Object-oriented database system has become increasingly popular and influential in the development of new generation database systems. This motivates the investigation of mechanisms for data mining in object-oriented databases. In this paper, we propose our first step towards knowledge discovery in object-oriented databases by extension of the attribute-oriented induction technique from relational databases to object-oriented databases. By the development of sophisticated generalization operators and generalization control mechanisms, attribute-oriented induction method can be successfully extended to knowledge discovery in object-oriented databases. Furthermore, we show that knowledge discovery will substantially enhance the power and flexibility of querying data and knowledge in object-oriented databases.

1 Introduction

With the rapid growth in the amount of information stored in databases, the development of efficient and effective tools for ***knowledge discovery in databases*** (*KDD*, or *data mining*) has become an increasingly important task in both database and machine learning researches [28, 6]. Since object-oriented database systems are popular and influential in advanced database applications [16, 22, 4], it is important to study the mechanisms for knowledge discovery in object-oriented databases (*OODBs*).

Object-oriented data models and systems [1, 2, 3, 12, 15] embody rich data structures and semantics in the construction of complex databases, such as complex data objects, class/subclass hierarchy, property inheritance, methods and active data, etc. This not only brings the power and flexibility to the system but also adds complexity to the implementations, including the development of knowledge discovery mechanisms [18].

There are different philosophical considerations on knowledge discovery in databases [7, 6, 29], which may lead to different methodologies in the development of KDD techniques. To simplify our discussion, the following assumptions are made as the first step in the development of mechanisms for knowledge discovery in OODBs.

*The research of the first author was supported in part by the Ministry of Education, Science and Culture of Japan under Scientific Research Grant-in-Aid, that of the second author was supported in part by a scholarship from the Ministry of Education, Science and Culture of Japan, and that of the third author was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada and the Centre for Systems Science of Simon Fraser University. The work by the second author was done during his visit to Simon Fraser University.

Assumption 1 *An OODB stores a large amount of information-rich, relatively reliable and stable data.*

Data in an OODB may be incomplete, redundant, incorrect, or highly dynamic in certain applications [29], which makes knowledge discovery a challenging task. The assumption on data stability and reliability facilitates the development of knowledge discovery mechanisms firstly in relatively simple environments and then evolution of the techniques step-by-step towards more complicated situations.

Assumption 2 *A knowledge discovery process is initiated by a user's learning request.*

Idealistically, one may expect that a knowledge discovery system will perform interesting discovery autonomously without human interaction. However, since learning can be performed in many different ways on any subset of data in the database, huge amount of knowledge may be generated from even a medium size database by unguided, autonomous discovery, whereas much of the discovered knowledge could be out of user's interests. In contrast, a command-driven discovery may lead to the guided discovery with a focus on the interested set of data and therefore represents relatively constrained search for the desired knowledge. Thus, command-driven discovery is adopted in this study.

Assumption 3 *Generalized rules are expressed in terms of high level concepts.*

Without concept generalization, discovered knowledge is expressed in terms of primitive data (data stored in the databases), often in the form of functional or multivalued dependency rules or primitive level integrity constraints. On the other hand, with concept generalization, discovered knowledge can be expressed in terms of concise, expressive and higher level abstraction, in the form of generalized rules or generalized constraints, and be associated with statistical information. Obviously, it is often more desirable for large databases to have rules expressed at concept levels higher than the primitive ones.

Assumption 4 *Background knowledge is generally available for knowledge discovery process.*

Discovery may be performed with the assistance of relatively strong background knowledge (such as conceptual hierarchy information, etc.) or with little support of background knowledge. Obviously, the discovery of conceptual hierarchy information itself can be treated as a part of knowledge discovery process. However, the availability of relatively strong background knowledge not only improves the efficiency of the discovery process but also expresses user's preference for guided generalization, which may lead to an efficient and desirable generalization process.

Following these assumptions, our mechanism for knowledge discovery in OODB can be outlined as follows. First, a knowledge discovery process is initiated by a learning request, which is usually in relevance to only a subset of data in an OODB. A data retrieval process is initiated to collect the set of relevant data. Second, generalization is performed on the set of retrieved data using the background knowledge and a set of generalization operators. Third, the generalized data is simplified and transformed into a set of generalized rules, which may facilitate query answering and many other applications.

The first step corresponds to the processing of a database query. Query processing in OODB is another subject of study, and many OODB query processing techniques have been developed recently [2, 14, 15, 27, 21]. Here we simply assume that the data set in relevance to the learning task is extracted efficiently by a data retrieval process without a detailed examination of query processing itself.

The second and third steps are the focus of this study. The KDD process is performed by extension of an attribute-oriented induction method which was first developed for knowledge discovery in relational databases [9, 10]. The method focuses on the generalization of individual attributes

without examination of the inter-relationships among different attributes at low concept levels. The generalization is performed by attribute-oriented concept tree ascension or applying other set-oriented generalization operators. The method integrates *learning-from-examples* techniques with database operations and substantially reduces the computational complexity of database learning processes.

The paper is organized as follows. In Section 2, generalization operators for complex data objects are examined systematically. In Section 3, generalization processes are studied with an emphasis on the control of generalization. In Section 4, a generalization algorithm is presented and analyzed in detail using some examples. In Section 5, the application of discovered knowledge is examined. The study is summarized in Section 6.

2 Generalization operators for complex objects

An OODB organizes a large set of complex data objects into classes which are in turn organized into class/subclass hierarchies with rich data semantics. Each object in a class is associated with (1) an object-identifier, (2) a set of attributes which may contain sophisticated data structures, set- or list- valued data, class composition hierarchies, multimedia data, etc. and (3) a set of methods which specify the computational routines or rules associated with the object class.

To facilitate the development of KDD mechanisms in OODBs, it is important to efficiently implement a relatively small set of generalization operators on which a large set of possible generalization operations can be constructed.

Formally, the generalization of a component p of an object O_i can be written in an abstract way as $Gen(O_i.p)$, where Gen is an abstract object generalization operator which can be transformed into a concrete operation based on the role of the component and the specific learning requirement. Moreover, the generalized component of an object can be further generalized by applying Gen again, which could be the same or different generalization operators compared with the one applied in the last generalization. If the generalization is performed by applying the same sequence of generalization operators on a component p of O_i, we should have

$$Gen^{n-1}(Gen(O_i.p)) \equiv Gen(Gen^{n-1}(O_i.p)). \quad (1)$$

For example, a person P_1's address can be generalized from a detailed address, such as the number of a street, into a higher leveled one, such as a street block, then a street, a district, a city, a province, a country, etc. when necessary by applying the generalization operator Gen several times, such as $Gen(Gen(\cdots Gen(P_1.address)\cdots))$.

2.1 Generalization of object identifiers

One of essential components of an OODB is object identifier (*oid*) which uniquely identifies objects. It remains unchanged over structural reorganization of data. Since objects in an OODB are organized into classes which in turn belong to certain class and subclass hierarchy, the generalization of objects may refer to their corresponding class and subclass hierarchy. Therefore, an object identifier can be first generalized into its corresponding lowest subclass names which can in turn be generalized into a higher level class/subclass name by climbing up the class/subclass hierarchy.

Suppose the object identifier of an object O_i is $O_i.oid$, the name of the lowest subclass that O_i belongs to is $C_{i,j}$, and the name of the superclass of a class $C_{i,k}$ is $C_{i,k-1}$, for all k $(1 \leq k \leq j)$. The generalization on the object identifier $O_i.oid$ for O_i can be represented as:

$$Gen(O_i.oid) = C_{i,j}. \quad (2)$$

$$\forall k(1 \leq k \leq j) \; Gen(C_{i,k}) = C_{i,k-1}. \quad (3)$$

2.2 Generalization of different kinds of attributes

An object in an OODB is described by a set of attributes and a set of methods. We study the generalization of a set of attributes in OODBs. The attribute value of an object could be a character, a fixed length character string, a numerical value, a structure, a class composition hierarchy, a set-valued or list-valued data, or even some unformatted data, such as text, map, image, voice or other forms of multimedia data. The generalization of different and complex kinds of attribute values is a challenging task.

Suppose that an object property (attribute value) of an object O_i is p_k, its minimum generalized concept is p_{k-1}, which in turn has the minimum generalized concept p_{k-2}, etc. A sequence of generalization on the object property p_k can be represented as:

$$\forall k(1 \leq k \leq j) \;\; Gen(p_k) \;\; = \;\; p_{k-1}. \tag{4}$$

2.2.1 Generalization on unstructured nonnumerical and numerical values

Ordinary unstructured nonnumerical and numerical attributes are the most popularly encountered attributes in databases.

Generalization on nonnumerical values may rely on the available concept hierarchies specified by DB designers, domain experts or users. Concept hierarchies represent necessary background knowledge which directs the generalization process. Different levels of concepts are often organized into a taxonomy of concepts. The concept taxonomy can be partially ordered according to a general-to-specific ordering. The most general concept (corresponding to *level 0*) is the null description (described by a reserved word "*ANY*"), and the most specific concepts correspond to the specific values of attributes in the database. 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.

For example, the address "*5235 East Hastings St., Burnaby, B.C. Canada*" can be generalized by specifying a partial order of the generalization sequence, such as "*5235* $\Rightarrow$ *the 5200 block* $\Rightarrow$ *East Hastings St.* $\Rightarrow$ *North Burnaby* $\Rightarrow$ *Burnaby* $\Rightarrow$ *Greater Vancouver Area* $\Rightarrow$ *British Columbia* $\Rightarrow$ *Western Canada* $\Rightarrow$ *Canada* $\Rightarrow$ *North America*". Such a generalization sequence is obtained by consulting the user-/expert- specified concept hierarchy. For example, one may specify a general rule on how to transform a street number, such as "*5235*", into a street block, such as "*the 5200 block*". Also, one may indicate that deleting the street number from a street or deleting the city name from a province is a step of generalization, etc. A generalization sequence can be implicitly stored in tuples. In many cases, a portion of generalization sequence can also be specified explicitly by users/experts or derived from the knowledge stored elsewhere. For example, "*British Columbia* $\Rightarrow$ *Western Canada*" should either be specified by experts or derived from geographical information.

Although a conceptual hierarchy could be stored or partially stored as data in a database, a stored hierarchy may often need to be modified or refined based on user's learning requirements and/or database statistics. For example, if the learning requirement is to analyze the birth place of the *students* of Simon Fraser University, the level 1 concepts should be: {*B.C, other_provinces_in_Canada, foreign*}. However, if it is to analyze the birth place of the *faculty* of the University, the appropriate level 1 concepts could be {*North_America, Europe, Asia, other_countries*}. Both concept hierarchies can be obtained by dynamic adjustment of a given concept hierarchy based on the analysis of the statistical distribution of the relevant data sets.

Generalization on numerical attributes can be performed similarly but in a more automatic way by the examination of data distribution characteristics [5]. It may not require any predefined concept hierarchies. For example, the ages of the graduate students in a university can be generalized according to relatively uniform data distribution into several groups, such as {*below 23, 23-26, 26-30, over 30*}. Appropriate names can be assigned to the generalized numerical ranges, such as {*very young, young,* $\cdots$} by users or experts to convey more semantic meaning.

2.2.2 Generalization on set-valued, list-valued or other structure-valued data

An attribute may contain a set of values with homogeneous or heterogeneous types. Typically, a set-valued data can be generalized in two ways: (1) generalization of each value in a set into its corresponding higher level concepts, or (2) derivation of the general behavior of a set, such as the number of elements in the set, the types or value ranges in the set, the weighted average for numerical data, etc. Notice that in the case of set-valued attribute generalization, the generalization operator $Gen(p_k)$ indicates that the input p_k can be a set of values and the output "$p_{k-1} = Gen(p_k)$" may also be a set of values. Moreover, the generalization can be performed by applying different generalization operators to explore alternative generalization paths. In this case, the result of generalization must be a heterogeneous set.

For example, the *hobby* of a person is a set-valued attribute which contains a set of values, such as {*tennis, hockey, chess, violin, nintendo*}, which can be generalized into a set of high level concepts, such as {*sports, music, computer_games*}, or into 5 (the number of hobbies in the set), or both, etc. Moreover, a *count* can be associated with a generalized value to represent how many elements are generalized to the corresponding generalized value, such as {*sports(3), music(1), computer_games(1)*}, where *sports(3)* indicates *three kinds of sports*, etc.

A set-valued attribute may be generalized into a set-valued or a single-valued attribute; whereas a single-valued attribute may also be generalized into a set-valued one if the "hierarchy" is a lattice or the generalization follows different paths. Further generalizations on such a generalized set-valued attribute should follow the generalization path of each value in the set.

A list-valued or a sequence-valued attribute can be generalized in a way similar to the set-valued attribute except that the order of the elements in the sequence should be observed in the generalization. Each value in the list can be generalized into its corresponding higher level concept. Alternatively, a list can be generalized according to its general behavior, such as the length of the list, the type of list elements, the value range, weighted average value for numerical data, or dropping unimportant elements in the list, etc. A list may be generalized into a list, a set or a single value. For example, a sequence (list) of data for a person's education record: "*((B.Sc. in Electrical Engineering, U.B.C., 1980), (M.Sc. in Computer Engineering, U. Maryland, 1983), (Ph.D. in Computer Science, UCLA, 1987)*" can be generalized by dropping less important descriptions (sub-attributes) of each tuple in the list, such as "*((B.Sc., U.B.C., 1980), ···)*", or by retaining only the most important tuple(s) in the list, such as "*(Ph.D. in Computer Science, UCLA, 1987)*".

Set- and list-valued attributes are simple structure-valued attributes. In general, a structure-valued attribute may contain sets, tuples, lists, trees, records, etc. and their combinations. Furthermore, one structure can be nested in another structure at any level. Similar to the generalization of set- and list-valued attributes, a general structure-valued attribute can be generalized in several ways, such as (1) generalize each attribute in the structure whereas maintain the shape of the structure, (2) flatten the structure and perform generalization on the flattened structure, (3) remove the low-level structures or summarize the low-level structures by high-level concepts or aggregation, and (4) return the type or an overview of the structure.

2.2.3 Aggregation and approximation as a means of generalization

Besides concept tree ascension and structured data summarization, aggregation and approximation should be considered as important means of generalization, which is especially useful for the generalization of attributes with large sets of values and complex structures, or multimedia data, etc.

Take spatial data as an example. It is desirable to generalize a detailed geographic map into clustered regions, such as business, residential, industry, or agricultural areas according to the land usage. Such generalization often requires the merge of a set of geographic areas by spatial DB operations, such as spatial union. Approximation is an important technique in such generalization. During the spatial merge, it is necessary not only to merge the regions of similar types within

the same general class but also to ignore some scattered regions with different types if they are unimportant to the study. For example, different pieces of land for different purposes of agricultural usage, such as vegetables, grain, fruits, etc. can be merged into one large piece of land by spatial merge (a means of aggregation). However, such an agricultural land may contain highways, houses, small stores, etc. If the majority land is used for agriculture, the scattered spots for other purposes can be ignored, and the whole region can be claimed as an agricultural area as an approximation. In this case, scattered small regions can be merged into large, clustered regions by spatial operators, such as *spatial_union, spatial_overlapping, spatial_intersection*, etc., and such spatial operators can be considered as generalization operators in spatial aggregation and approximation.

2.2.4 Generalization on multimedia data

A multimedia database may contain variable-length text, graphics, images, maps, voice, music, and other forms of audio/video information. Such multimedia data are typically stored as sequence of bytes with variable lengths, and segment of data are linked together for easy reference. Generalization on multimedia data can be performed by recognition and extraction of the essential features and/or general patterns of such data.

There are many ways to extract the essential features or general patterns from segments of multimedia data. For an image, the size or color of the image, the major regions in the image can be extracted by aggregation and/or approximation (as discussed above). For a segment of music, its melody can be summarized based on the approximate patterns that repeatedly occur in the segment and its style can be summarized based on its tone, tempo, and major musical instruments played, etc. For an article, its abstract or general organization such as the table of contents, the subject and index terms frequently occurring in the article, etc. may serve as generalization results. In general, it is a challenging task to generalize multimedia data to extract the interesting knowledge implicitly stored in the data. Further research should be devoted to this issue.

2.3 Generalization on inherited and derived properties

OODBs are organized into class/subclass hierarchies. Some attributes or methods of an object class are not explicitly specified in the class itself but are inherited from its higher level classes. Some OODB systems may allow the properties to be inherited from more than one superclass (called *multiple inheritance*) when the class/subclass "hierarchy" are organized in the shape of a lattice. The inherited properties of an object can be derived by query processing in the OODB. From the knowledge discovery point of view, it is unnecessary to distinguish which data are stored within the class and which are inherited from its superclass. As long as the set of relevant data are collected by query processing, the KDD process will treat the inherited data in the same way as the data stored in the object class and perform generalization accordingly.

Method is another important component of OODBs. Many behavioral data of objects can be derived by application of methods. Since a method is usually defined by a computational procedure/function or by a set of deduction rules, it is difficult to perform generalization on the method itself unless the generalization of the method is clearly understood by application programmers and is coded as a new method which directly performs the required generalization. In general, the generalization on the data derived by method application should be performed in two steps: (1) deriving the task-relevant set of data by application of the method and, possibly, also data retrieval; and (2) performing generalization by treating the derived data as the existing ones.

3 Control of Generalization Processes

Task-relevant data can be viewed as examples for learning processes. Undoubtedly, *learning-from-examples* [19, 8] should be an important strategy for knowledge discovery in databases. Most *learning-from-examples* algorithms partition the set of examples into *positive* and *negative* sets and

perform *generalization* using the positive data and *specialization* using the negative ones [19]. Unfortunately, a relational database does not explicitly store negative data, and thus no explicitly specified negative examples can be used for specialization. Therefore, a database induction process relies mainly on generalization, which should be performed cautiously to avoid over-generalization.

A knowledge discovery process applies a sequence of generalization operators to a set of data to generate a new set of data. A set of data can be viewed as a class from the view of OODB. The class which is fed into a generalization operator for generalization is called the ***working class***, $\mathcal{W}$, with the initial one (the set of task-relevant data) called the ***initial working class***, $\mathcal{W}_0$. The class generated by the application of a generalization operator is called the ***resulting class***, $\mathcal{R}$.

An attribute-oriented induction method, developed in the study of knowledge discovery in relational databases [9, 10], can be extended to OODBs. The method is briefly outlined as follows. First, a set of generalization operators are selected and applied to an attribute in the working class without considering the inter-relationships among different attributes before the concepts in each attribute are generalized to a desired level. The reason to ignore the inter-relationships among different attributes at an early stage of generalization is that such inter-relationships, if considered at an early stage, would have to be expressed at an undesirably low level in large numbers. This cannot lead to elegant generalized rules to be expressed at a high level and in concise terms. Attribute-oriented induction, which considers the attribute inter-relationships only when the data has been generalized into a relatively small set, will substantially reduce the computational complexity of a database learning process.

Formally, a generalization operator *Gen* can be applied to an attribute a_i on every object in a working class $\mathcal{W}_k$, resulting in a new generalized class $\mathcal{R}_k$. Thus, a **database generalization operator**, *DBGen*, is introduced which applies the object generalization operator *Gen* to an attribute a_i of *every* object in the working class. That is,

$$\mathcal{R}_k = DBGen(\mathcal{W}_k, a_i) \quad = \quad \{o' : o \in \mathcal{W}_k \wedge o'.a_i = Gen(o.a_i) \wedge \forall (j \neq i)\ o'.a_j = o.a_j\} \qquad (5)$$

For example, for a working class $\mathcal{W}_0 = Person$ in a university database, $DBGen(\mathcal{W}_0, address)$ derives a resulting class $\mathcal{R}_0$ with one-step generalization on the attribute "*address*" (e.g., from street number to stree block) and all the other attributes unaltered.

A knowledge discovery process can be viewed as the application of a sequence of database generalization operators, *DBGen*, on different attributes until the resulting class contains a small number of generalized objects which can be summarized as a concise, generalized rule in high-level terms.

3.1 Basic strategies for attribute-oriented induction

In general, we have the following basic techniques for attribute-oriented induction [9, 10] in OODBs.

Technique 1 (Data focusing) *Generalization should be performed only on the set of data which are relevant to the learning request.*

Technique 2 (Attribute removal) *If there are a large set of distinct values in an attribute in the working class, but there is no generalization operator on the attribute, the attribute should be removed from the working class.*

Technique 3 (Attribute generalization) *If there are a large set of distinct values in an attribute in the working class, but there exists a set of generalization operators on the attribute, a generalization operator should be selected and applied to the attribute at every step of generalization.*

As a result of generalization, different objects may be generalized to equivalent ones where two (generalized) objects are **equivalent** if they have the same corresponding attribute values without considering their *object identifiers* and a special internal attribute **count**, which registers the number of objects in the initial working class that are generalized to the object in the current resulting class. Notice that a *generalized object*, though has its own new *oid* (*object identifier*) in an OODB, is a

carrier of the general properties of a set of initial objects because the original object identifier has been generalized into a class or superclass name. The *count* accumulated in the generalized class incorporates quantitative information in the learning process.

Technique 4 (Count propagation) *The value of the count of an object should be carried to its generalized object, and the count should be accumulated when merging equivalent objects in generalization.*

Technique 5 (Attribute generalization control) *Generalization on an attribute a_i is performed until the concepts in a_i has been generalized to a desired level, or the number of distinct values in a_i in the resulting class is no greater than a prespecified threshold.*

Notice that *the threshold which controls the attribute generalization* is called **attribute threshold** which is usually a small number (often between 2 to 10) that can be specified explicitly by a user/expert or be built in the system as a default.

Theorem 1 *The above five generalization techniques are correct and necessary for the extraction of generalized rules from databases.*

Proof sketch. Technique 1 is obvious since only the task-relevant set of data need to be studied. An attribute-value pair represents a conjunct in a generalized rule, the 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 generalization operator for it, the attribute should be removed. Thus, we have Technique 2 which corresponds to the generalization rule, *dropping conditions*, in *learning-from-examples* [19]. The generalization of an attribute value using a selected generalization operator makes the object covers more cases than the original one and thus generalizes the concept. Thus, we have Technique 3 which corresponds to the generalization rule, *climbing generalization trees*, in *learning-from-examples* [19]. Technique 4 is based on the merge of identical tuples. Technique 5 is based on the desirability of representation of each attribute at its desired level. Thus, we have the theorem. □

The application of a database generalization operator *DBGen* on a *working class* results in a more general *resulting class*, which in turn becomes the working class in the next round of database generalization. Such a generalization process proceeds until the concept in every attribute in the resulting class has reached to a desired concept level, or the number of distinct values in every attribute is no greater than its attribute threshold. The generalized resulting class so obtained is called a **prime generalized class**.

3.2 Generalized rule extraction

Since the above induction process enforces only attribute generalization control, the prime generalized class so extracted may still contain a relatively large number of generalized objects. Two alternatives can be developed for the extraction of generalized rules from a prime generalized class: (1) further generalize the class to derive a **final generalized class** which contains no more objects than a prespecified *class threshold*, and then extract the *final generalized rule*; and (2) directly extract **generalized feature table** and present feature-based multiple rules.

Alternative 1 is based on the following Technique 6. The interestingness of the final generalized rule relies on the selection of the attributes to be generalized and the selection of generalization operators. Such selections can be based on data semantics, user preference, generalization efficiency, etc. A more detailed discussion is provided in the *discussion* section.

Technique 6 (Class generalization control) *Generalization on a prime generalized class is performed until the number of distinct generalized objects in the resulting class is no greater than a prespecified class threshold.*

Alternative 2 takes the set of generalized objects and maps them into a generalized feature table. Based on the generalized feature table, multiple generalized feature-based rules can be presented. The algorithm for the derivation of the generalized feature table is similar to that for relational databases [10] and is exemplified in the next section.

4 Generalization Algorithm and Example

The above discussion can be summarized into the following generalization algorithm which extracts generalized characteristic rules in an OODB based on a user's learning request, where a **characteristic rule** is an assertion which characterizes the concepts satisfied by all or a majority number of the examples in the task-relevant data set.

Algorithm 1 (Basic attribute-oriented induction in OODB) *Discovery of a set of generalized characteristic rules in an OODB based on a user's learning request.*

Input. (i) An OODB DB, (ii) $Gen(a_i)$, a set of concept hierarchies or generalization operators on attributes a_i, and (iii) T, a *class threshold*, and T_i, a set of *attribute thresholds* for attributes a_i.

Output. A *characteristic rule* based on the learning request.

Method.

1. Derive the *initial working class*, $\mathcal{W}_0$, i.e., collect the set of task-relevant data by an OODB query based on the learning request.
2. Derive the *prime generalized class*, $\mathcal{R}_p$, by performing a sequence of attribute-oriented induction, $\mathcal{I}(k)$, for $(0 \leq k \leq p)$ on the initial working class $\mathcal{W}_0$. The induction $\mathcal{I}(k)$, performs $DBGen$ on the working class $\mathcal{W}_k$ and generates a resulting class $\mathcal{R}_k$. The resulting class $\mathcal{R}_k$ is taken as the working class $\mathcal{W}_{k+1}$ in the next induction $\mathcal{I}(k+1)$. The processing detail is described as follows.
   begin
   for each relevant attribute (method) a_i in $\mathcal{W}_k$ **do** {
   if a_i should be removed (i.e., there are a large set of distinct values in a_i in $\mathcal{W}_k$, but there is no generalization operator on a_i)
   then remove a_i
   else if a_i is not at a desired level
   then repeatedly apply: (1) $\mathcal{I}(k)$: $\mathcal{R}_k := DBGen(\mathcal{W}_k, a_i)$, (2) $\mathcal{W}_{k+1} := \mathcal{R}_k$, and (3) $k := k + 1$, until the attribute a_i is generalized to a desired level. }
   Merge equivalent objects in $\mathcal{R}_p$ with *count* accumulated.
   — Comments: Generalization on each attribute a_i can be implemented efficiently by (1) collecting the distinct a_i values in the working class, (2) computing the minimum desired level L, and (3) generalizing the attribute to this level L by replacing each value in a_i's with its corresponding superordinate concept in H_i (the concept hierarchy for a_i) at level L (which may be obtained by a sequence of Gen's).
   end
3. Presentation of generalized rules.
   - Determine which of the two alternatives: (1) *final generalized class*, or (2) *generalized feature table*, should be chosen in the presentation of generalized rules. This can be predetermined by experts or determined by interaction with users.
   - **If** alternative 1 is chosen,
     then further generalization (similar to Step 2) is performed on the prime generalized class by selection of certain attributes for generalization until the number of distinct generalized objects is no greater than the *class threshold*, T. The final generalized class can be mapped to a rule (which is the *final generalized rule*) for output.

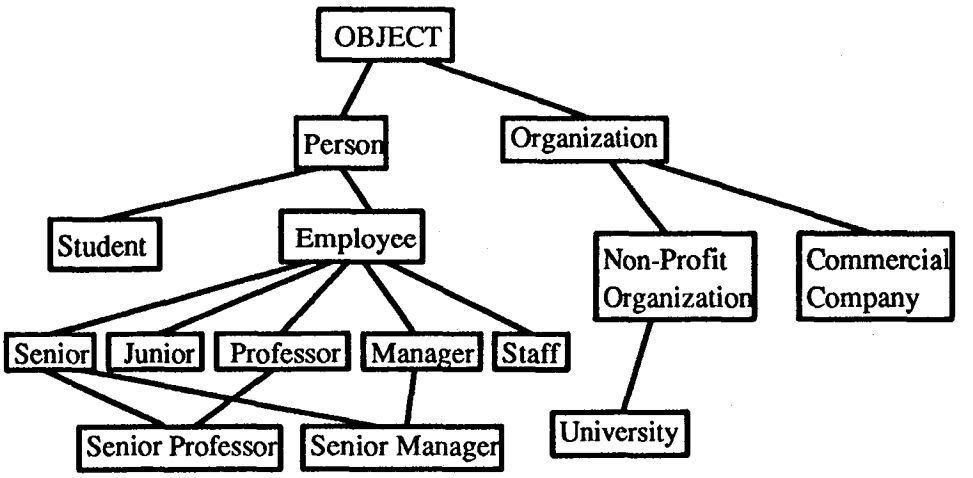

Figure 1: Class hierarchy in OODB.

- **If** alternative 2 is chosen,
  then the prime generalized class is mapped into a *generalized feature table*, which can be further mapped into a set of generalized rules for output. □

Step 1 of the algorithm is essentially an OODB query whose processing efficiency depends on a particular query processing algorithm. We have the following theorem for the processing efficiency of Steps 2 & 3.

Theorem 2 *The worst-case time complexity of Steps 2 & 3 in Algorithm 1 is* $O(n \log(n))$ *where* n *is the number of data objects relevant to the learning request.*

Proof sketch. The cost of Step 2 is dominant in Steps 2 & 3 processing since Step 3 works on a much smaller set of objects than Step 2 but using a similar algorithm. For every task-relevant attribute, (1) the collection of attribute values in the working class takes at most $O(n \log(n))$ (retrieval of every object in the class takes $O(\log(n))$ using an appropriate indexing structure such as a B+-tree), (2) the redundant value elimination involves sorting which takes $O(n \log(n))$ for n object values, (3) computing minimum desired level involves generalization of these object using a sequence of generalization operators, and total $O(n) \times l \times c_g$ generalization will be performed, where l is the number of levels and c_g is the cost of execution of each generalization operator, and (4) redundant value elimination takes at most $O(n \log(n))$. Summing them together, the generalization for each attribute takes at most $O(n \log(n))$ time. Since only a small, constant number of attributes are relevant to a learning task, the worst case time complexity should be $O(n \log(n))$. □

Example 1. Let an OODB consist of a set of classes *Person*, *Organization*, etc. and their associated subclasses. A portion of class/subclass hierarchy is shown in Fig. 1, where a subclass *senior_professor* is a subclass of both *senior_employee* and *professor* (i.e., *multiple inheritance*). A portion of the conceptual hierarchy for the attribute *address* is shown in Fig. 2, where the concepts higher than *Canada* such as *North_America*, and that lower than the *city* level such as *district, street, block, etc.* are not presented in the figure.

Let the learning task be to discover a characteristic rule in relevance to *housesize, residential area* and *salary* for those with a Ph.D. degree, associated with U.B.C, 40 (years-old) or over, and driving Japanese cars.

The learning task can be represented in the following query [10] in the syntax similar to XSQL [14].

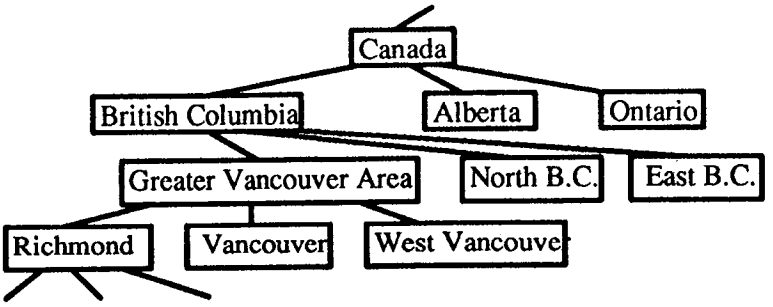

Figure 2: Concept hierarchy in OODB.

```
LEARN Characteristic rule
FROM  Person P
WHERE P.Age >= 40 and P.Car.Maker = 'Japan'
      and 'Ph.D' in P.Education and P.Workplace.Name = 'U.B.C.'
IN RELEVANCE TO P.Name, P.Home.Housesize, P.Salary, P.Home.Address
```

Following Algorithm 1, the learning request can be processed as shown below.

Step 1 Collect objects that are related to the learning task. One collected object is supposed to be as follows. Notice that some of attributes in the object could be defined by methods or be inherited from its superordinate classes. Furthermore, the query processing will involve checking against "House_Object", "Workplace_Object" and "Car_Object" to satisfy the query condition.

Object[1]

ObjectID:	02EREV	;String value is given by system
Name:	Alex Fleming	;String value
Home:	<House_ObjectID>	;House_Object has "Housesize", "Address", "Price" and so on.
Education:	(···, (Ph.D. in Computer Science, UCLA, 1987))	;Set-data values
Workplace:	<Workplace_ObjectID>	;Workplace_Object has "Name", "Address" and so on.
Salary:	$73,854	;Real value
Birthdate:	23,March,1950	;String value
Age:	Method(Birthdate,Today)	;Method
Face:	<Bitmap data>	;Multimedia data

⋮

Step 2 During the generalization process, the attribute "P.Name" is removed (even if it were in relevance to the learning query) since it has no superordinate concepts except "*ANY*". With attribute generalization control (assuming that every attribute threshold is 4 by default), the generalization derives the following high level concepts for the task-relevant attributes: {*Richmond, West Vancouver, Vancouver*} for *Address* (at the concept level of *city*), {*senior professor, senior manager*} for object identifier (along the *class/subclass hierarchy*), {*Big, Medium, Small*} for *Housesize*, and {*M (medium), H (high), VH (very high)*} for *Salary*.

Since the generalized objects contain a small number of distinct attribute values, there are high possibilities that some of them are equivalent and can be merged together. By merging equivalent generalized objects, the prime generalized class is extracted. One generalized object in the prime generalized class is shown below.

Generalized_Object[1]

Class: Senior professor
ObjectID: 60@WUH
Housesize: Big
Address: Richmond
Salary: High
Count: 64

Step 3 Suppose that alternative 2 is chosen in Step 3 of Algorithm 1. The prime generalized class is mapped into a generalized feature table as shown in Table 1.

		senior professor				senior manager				Total
		M	H	VH	Total	M	H	VH	Total	
Big	Richmond	0	64	103	167	0	16	24	40	207
	West Vancouver	0	29	46	75	0	9	9	18	93
	Vancouver	0	11	39	50	0	4	8	12	62
	Total	0	104	188	292	0	29	41	70	362
Medium	Richmond	1	42	20	62	0	12	1	13	75
	West Vancouver	0	21	62	83	0	3	18	21	104
	Vancouver	0	12	32	44	1	2	5	7	51
	Total	1	75	114	190	1	17	23	41	231
Small	Richmond	0	1	0	1	1	0	0	1	2
	West Vancouver	1	0	0	1	0	1	0	1	2
	Vancouver	0	2	0	2	0	0	0	0	2
	Total	1	3	0	4	1	1	0	2	6
Total		2	182	302	486	2	47	64	113	599

Table 1: A generalized feature table for the learning task.

Interesting rules can be extracted from this feature table, such as the following: *Those who satisfy the query condition are either senior professors (about 81%) or senior managers (about 19%). About 97% of them earn high or very high salary, and about 60% of them are living in big houses in these three cities: Richmond, Vancouver and West Vancouver, etc.* □

5 Discussion

5.1 Variations to the basic induction algorithm

The basic attribute-oriented induction method presented above is only for the discovery of general characteristic rules for a set of task-relevant data. Similar to knowledge discovery in relational databases [9], the technique can be extended to the discovery of different kinds of rules in OODBs, such as discriminant rules, data evolution regularities, approximate rules, etc. **A discriminant rule** is an assertion that discriminates a concept of the class being learned (called the *target class*) from other classes (called the *contrasting classes*). **Data evolution regularity** reflects the general trend of changes in data contents in the database over time. An **approximate rule** is the rule which represents the characteristics of a *majority* number of facts in the database, which is especially useful when databases contain noisy data and exceptional cases.

As an example, we briefly outline how to extend the above method to the discovery of discriminant rules. To distinguish two sets of objects with different properties, it is necessary to first collect the task-relevant objects into two classes: the *target class* and the *contrasting class*, and then perform generalization *synchronously* on the two classes. Once the concepts are generalized to the *same* high level, general behaviors can be compared between the two classes and the discriminative behaviors can be extracted as discriminant rules.

Another important issue is the control of a discovery process. When every relevant attribute has been generalized to a relatively high concept level (especially when it reaches the desired concept level), the selection of attributes for further generalization becomes subtle. Criteria for selection of one attribute for generalization instead of another should be based on the concern of data and query semantics, user-preference, efficiency, etc. For example, if certain attribute becomes a focus of study, the selection of that attribute for progressive generalization should often be conservative in order to observe its regularity with the changes of other attributes.

Semantic meaning and efficiency should be the major concerns in the selection of generalization of complex data objects. In an OODB containing both spatial and nonspatial data, generalization on spatial data may sometimes suffer from efficiency or meaningful semantic interpretation problems. It is recommended to first generalize nonspatial data since it is relatively efficient to generalize well-defined nonspatial data, and the semantics of such generalization is usually clear. Such a generalization and merge of objects with similar nonspatial general properties may guide the merge of spatial objects and trigger the spatial generalization and approximation operation, such as *spatial_union*. For example, the merge of agricultural land with different crops, fruits, and vegetables can be performed by first generalizing the values in the attribute "*land_usage*" into "*agriculture*", and then merging the regions with similar usage. Such a high-level semantic-driven spatial generalization will capture more meaning and often lead to better efficiency than blindly generalizing spatial data according to its spatial data structures, such as quad-trees or R-trees [26].

Another issue on the control of a generalization process is the use of different generalization operators on the same attribute. If there is more than one generalization operator available in the system, one may selectively apply some generalization operator. Such a selection of generalization operators may also be based on data semantics, user/expert instructions, execution history, and processing efficiency. When it is not easy to make such a selection, one may proceed along several generalization paths in parallel until later stage because promising regularities may be discovered by pursuing different generalization paths.

5.2 Application of discovered knowledge

The regularities discovered by generalization in OODBs will be useful at querying data and knowledge implicitly or explicitly stored in OODBs, improving query/transaction processing and other system performance for OODB systems, and helping the design and evolution of OODBs. Because of the limited space, only the first aspect, ***the application of discovered knowledge to querying data and knowledge in OODBs***, will be examined in this subsection.

Example 2. Let the database and conceptual hierarchies be the same as that presented in Example 1. We examine how conceptual hierarchies and knowledge discovery tools may help querying data and knowledge.

First, generalization may substantially increase the power and flexibility of querying OODBs. For example, to find everyone who has graduate degree, works in U.B.C., living in a big and expensive house in West Vancouver. One may formulate a query containing high level concepts, such as, *P.home.house_size = "big", P.home.house_value = "expensive", graduate_degree in P.education*, etc. Such concepts cannot be expressed in a query without the help of concept generalization since database stores only the dimensions or the value of a house but not high-level concepts like "*big*", or "*expensive*". With the availability of concept hierarchies and generalization operators, the concrete data in the database, such as the house value can be generalized into "expensive", "cheap", etc. by applying a sequence of *Gen* operations.

Second, with the availability of generalization tools, queries can be answered in a more general and interesting manner [11, 20]. Suppose the learning request of Example 1 is rewritten as a query. That is, the request is to "**select** P.Name, $\cdots$ **from** Person P **where** $\cdots$" which is to find the names, etc. of all the persons who satisfy the same condition as the learning request in Example 1. A strict query answering will print the names and other inquired information for every 599 persons, which

could be very boring to users. An intelligent way to answer the query is to summarize the information and print the general characteristics as demonstrated by Example 1. The detailed information for 599 persons will be printed only when the user is not satisfied with the general answers. Obviously, high-level answers are more attractive to most users.

Third, many different kinds of rules and generalities can be summarized using knowledge discovery tools described in this paper which may substantially increase the usability of the data stored in the database and discover important regularities for many applications. For example, one may find good bargains on house purchasing after examination of high-level regularities among *house_location*, *house_value*, *house_price*, etc. in a database. □

6 Conclusions

In this paper, an initial step is proposed and investigated for knowledge discovery in OODBs. First, sophisticated generalization operators are examined for generalizing and handling complex data objects, such as structured data, methods, inherited data, object identifiers, class/subclass hierarchies, multimedia data, etc. Second, generalization control mechanisms are developed by extensions to the attribute-oriented induction method originally designed for knowledge discovery in relational databases. With further development of generalization operators and generalization control mechanisms, knowledge discovery can be performed in OODBs efficiently and effectively based on our study and complexity analysis of the developed techniques. Finally, we demonstrate that the availability of generalization operators and knowledge discovery tools will substantially enhance the power and increase the flexibility of data and knowledge retrieval in OODBs.

There are many research issues on knowledge discovery in OODBs. The construction of efficient and effective generalization operators for complex structured or unstructured data, such as hypertext and multimedia data, is an important but unsolved issue. The control of generalization directions for objects in the same class but with different features in order to merge the common (generalized) features is another issue which should be studied further. Another interesting issue is the construction of a multi-resolution model for OODBs [25], which will help browse OODB contents and answer interesting queries at high concept levels. Finally, software development and experimentation should be performed on the proposed mechanisms for KDD in OODBs to verify and improve the proposed technique and compare it with other related, promising proposals [13, 24, 23, 17, 29].

References

[1] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database systems manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio (editors), *Deductive and Object-Oriented Databases*, pages 223–240. Elsevier Science, 1990.

[2] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The story of O2.* Morgan Kaufmann, 1992.

[3] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. Classic: A structural data model for objects. In *Proc. 1989 ACM-SIGMOD Conf. Management of Data*, pages 58–67, Portland, Oregon, June 1989.

[4] R.G.G. Cattel. *Object Data Management: Object-Oriented and Extended Relational Databases.* Addison-Wesley, 1991.

[5] D. Fisher. Improving inference through conceptual clustering. In *Proc. 1987 AAAI Conf.*, pages 461–465, Seattle, Washington, July 1987.

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

[7] B. R. Gaines and J. H. Boose. *Knowledge Acquisition for Knowledge-Based Systems.* London: Academic, 1988.

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

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

[10] J. Han, Y. Cai, N. Cercone, and Y. Huang. Dblearn: A knowledge discovery system for databases. In *Proc. 1st Int'l Conf. on Information and Knowledge Management*, pages 473–481, Baltimore, Maryland, Nov. 1992.

[11] J. Han, Y. Huang, and N. Cercone. Intelligent query answering by knowledge discovery techniques. In *IEEE Trans. Knowledge and Data Engineering*, (to appear), 1993.

[12] K. Higa, M. Morrison, J. Morrison, and O. R. Sheng. An object-oriented methodology for knowledge base/database coupling. *Comm. ACM*, 35:99–113, June 1992.

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

[14] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented database. In *Proc. 1992 ACM-SIGMOD Conf. Management of Data*, pages 393–402, San Diego, CA, June 1992.

[15] W. Kim. *Introduction to Object-Oriented Databases.* The MIT Press, 1990.

[16] W. Kim and F. H. Lochovsky. *Object-Oriented Languages, Applications, and Databases.* Addison-Wesley, 1989.

[17] W. Kloesgen. Patterns for knowledge discovery in databases. In *Proc. ML-92 Workshop on Machine Discovery*, pages 1–10, Aberdeen, Scotland, July 1992.

[18] M. Manago and Y. Kodratoff. Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro and W. J. Frawley (editors), *Knowledge Discovery in Databases*, pages 289–306. AAAI/MIT Press, 1991.

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

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

[21] J. Orenstein, S. Haradhvala, B. Margulie, and D. Sakahara. Query processing in the ObjectStore database system. In *Proc. 1992 ACM-SIGMOD Conf. Management of Data*, pages 403–412, San Diego, CA, June 1992.

[22] K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong. *Intelligent DataBasese.* John Wiley & Sons, 1989.

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

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

[25] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.

[26] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[27] G.M. Shaw and S. B. Zdonik. A query algebra for object-oriented databases. In *Proc. 6th Int. Conf. Data Engineering*, pages 154–162, Los Angeles, CA, February 1990.

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

[29] J. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley (editors), *Knowledge Discovery in Databases*, pages 31–54. AAAI/MIT Press, 1991.