Efficient Algorithms for Attribute-Oriented Induction

Hoi-Yee Hwang and Wai-Chee Fu

Department of Computer Science
Chinese University of Hong Kong
Shatin, Hong Kong
hyhwang@cs.cuhk.hk, adafu@cs.cuhk.hk

Abstract

Data mining or knowledge discovery in databases is the search for relationships and global patterns that exist but are hidden in large databases. Many different methods have been proposed and one of them is the attribute-oriented induction method. In this method, domain knowledge in the form of concept hierarchies helps to generalize the concepts of the attributes in the database relations. This approach has been generalized to the rule-based attribute-oriented induction. The time complexity of the original algorithms is given by O(N log N), where N is the number of relevant tuples in the database. In this paper, we make use of the static property of the database schema and the concept hierarchies to derive more efficient algorithms. Given that the concept hierarchies and the resulting knowledge are small in size compared to the database, the complexity of our algorithm is O(N). The amount of disk I/O is decreased by O(log N) times compared to the previous methods. We believe that this improvement in performance will give extra power to the attribute-oriented method.

1. Introduction

Data mining is the search for relationships and global patterns that exist in large databases, but are 'hidden' among the vast amounts of data [Frawley, Piatetsky-Shapiro & Matheus 1991]. Many different methods for data mining, or knowledge discovery in databases, have been proposed in the past. Overview of this area can be found in [Agrawal 1994, Piatetsky-Shapiro & Frawley 1991]. Some recent works include [Agrawal & Srikant 1994, Faloutsos & Lin 1995, Park,Chen & Yu 1995].

In [Cai, Cercone & Han 1991, Han, Cai & Cercone 1992, Han, Cai & Cercone 1993], an attribute-oriented induction method for data-driven discovery of quantitative rules in relational databases is presented and a database learning system DBLEARN has been constructed based upon this approach [Han et al. 1992]. It uses domain knowledge to generate descriptions for predefined subsets in a relational database. The method integrates learning-from-examples techniques with database operations and extracts generalized data from actual data in databases. This attribute-oriented approach uses the concept hierarchy to direct the learning process. In the attribute-oriented induction process, lower level concepts in a concept tree or lattice are generalized to higher level concepts. The generalization algorithm can be well integrated with database operations, since generalization operations are set-oriented, and both data and knowledge are represented as relational tables.

It has been shown that the complexity of this attribute-oriented approach is O(NlogN) [Cheung, Fu & Han 1994, Han, Cai & Cercone 1993], where N is the size of the initial relation. In this paper, we would enhance this performance. It is found that, as soon as the concept hierarchies are given, the generalization path of each attribute in each tuple of the database can be found. So we can set a path id for each attribute concept in the database. The generalization step is made much more efficient and an improved algorithm of O(N) is proposed. We also replace the algorithm in [Cheung, Fu & Han 1994] for the rule-based attribute-oriented approach by this method. With the help of path id, backtracking is eliminated and an efficient algorithm of O(N) is derived.

The paper is organized as follows. Section 2 gives a brief review on the original attribute-oriented induction approach. Terminology and definitions are introduced in Section 3. Preprocessing work will be stated in Section 4. In Section 5, an improved generalization algorithm for the attribute-oriented induction is proposed, and the complexity of this algorithm is discussed in Section 6. Generalization procedure for rule-based attribute-oriented induction approach is presented in Section 7. A conclusion will be given in Section 8.

2. Original Attribute-Oriented Approach

In [Cai, Cercone & Han 1991, Han, Cai & Cercone 1992, Han, Cai & Cercone 1993], an attribute-oriented induction method for data-driven discovery of quantitative rules in relational databases is presented. It uses domain knowledge to generate descriptions for predefined subsets of a relational database. This attribute-oriented approach uses the concept hierarchy to direct the learning process.

A concept hierarchy is related to a specific attribute and is partially ordered according to a general-to-specific ordering. The most general point in the hierarchy is the null description (ANY), while the most specific points correspond to the specific values of an attribute in the database.

For example, assume that a university student database has the following schema.

Student(Name, Status, Sex, Age, GPA)

The concept tree table for Student is shown in Figure 1, the concept tree of the attribute Status will be the one shown in Figure 2.

{freshman} ⟶ undergraduate {sophomore} ⟶ undergraduate
{junior} ⟶ undergraduate {senior} ⟶ undergraduate
{M.A.} ⟶ graduate {M.S.} ⟶ graduate
{Ph.D.} ⟶ graduate
{undergraduate, graduate} ⟶ ANY(Status)
{0.0-1.99} ⟶ poor {2.0-2.99} ⟶ average
{3.0-3.49} ⟶ good {3.5-4.0} ⟶ excellent
{poor, average} ⟶ weak {good, excellent} ⟶ strong
{weak, strong} ⟶ ANY(GPA)
{M, F} ⟶ ANY{Sex)
{16-25} ⟶ 16_25 {26-30} ⟶ 26_30
{16_25, 26_30} ⟶ ANY(Age)

Figure 1 Concept tree table for a university student database

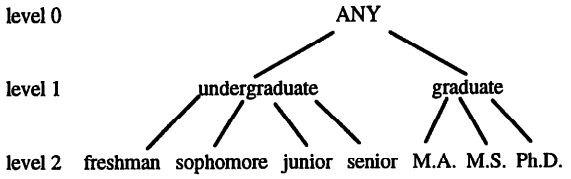


Figure 2 Concept tree for Status

A relation which represents intermediate (final) learning results is called an **intermediate (final) generalized relation**. In a generalized relation, some or all of its attribute values are generalized data, that is, nonleaf nodes in the concept hierarchies. An attribute in a (generalized) relation is at a desirable level if it contains at most a small number of distinct values in the relation. This small number is specified by the user as a desirable **attribute threshold**.

A set of basic principles for the attribute-oriented induction in relational databases is summarized as follows.

1. Generalization should be performed only on the set of data which is relevant to the learning task.
2. Generalization should be performed on the smallest decomposable components (or attributes) of a data relation.
3. Attribute removal: If an attribute has too many distinctive values and there is no higher level concept provided for further generalization, it should be removed from the relation.
4. Concept tree ascension: For an attribute in an intermediate relation, if its values can be generalized to higher level concepts in the concept tree of the attribute, all values of the attributes are replaced by the higher level concepts. Outcome of the ascension is a generalized relation.
5. Vote propagation: Vote of a generalized tuple indicates the number of tuples in the initial relation that are generalized to this tuple. The value of the vote of a tuple is carried to its generalized tuple and the votes should be accumulated when merging tuples.
6. Attribute threshold control: For an attribute, if the number of its distinct values in an intermediate relation is still larger than its desirable attribute threshold, further generalization on this attribute should be performed.

By applying the above principles, an initial relation would be reduced to a generalized relation call **prime relation**. This prime relation has a small number of distinct values (less than or equal to the attribute threshold). This prime relation may need to be generalized further to produce the final relation. Two additional principles are used to complete the Attribute-Oriented induction process.

1. Generalization threshold control: If the number of tuples in a generalized relation is larger than the **generalization relation threshold**, further generalization should be performed.
2. Rule formation: A tuple in the final relation is transformed to conjunctive normal form, and multiple tuples are transformed to disjunctive normal form.

3. Terminology and definition

Suppose the database we work on has n attributes. A concept tree T_i is given for each attribute A_i, for i = 1,...,n. For each T_i, the root is denoted by ANY and assume that each T_i is a balanced tree.

Definition 1 For an attribute A, let T be the concept tree of A. For each leaf node a of the concept tree T, we call the path from a to the root ANY a **generalization path**.

Definition 2 Two paths are said to be **equivalent** if they pass through the same non-empty set of nodes. Two paths are **distinct** if they are not equivalent.

4. Preprocess for generalization

We shall make use of the distinct paths. Assume that there are m_i distinct paths for attribute A_i, we can label the distinct paths by $\{1,2,...,m_i\}$, we call this the **path id** of the path. Each ground value a of attribute A_i has a unique path to the root ANY whose path id is r, where $1 \le r \le m_i$.

The entire database can be transformed to a Path relation which contains only path id of each ground value. For example, if the generalization relation threshold g is 6 and the concept tree table and the corresponding path id's for a university student database is as shown in Figure 3, the Initial relation of Table 1 will be transformed to the Path relation shown in Table 2. The attribute Name is removed as we found that there are a large number of distinct names in the initial relation and there is no concept at a higher level to generalize these names.

attribute	path id	generalization path
Name	removed	no
Status	1	freshman ⟶ undergraduate ⟶ ANY
	2	sophomore ⟶ undergraduate ⟶ ANY
	3	junior ⟶ undergraduate ⟶ ANY
	4	senior ⟶ undergraduate ⟶ ANY
	5	M.A. ⟶ graduate ⟶ ANY
	6	M.S. ⟶ graduate ⟶ ANY
	7	Ph.D. ⟶ graduate ⟶ ANY
Sex	1	M ⟶ ANY
	2	F ⟶ ANY
Age	1	{16-25} ⟶ 16_25 ⟶ ANY
	2	{26-30} ⟶ 26_30 ⟶ ANY
GPA	1	{0.0-1.99} ⟶ poor ⟶ weak ⟶ ANY
	2	{2.0-2.99} ⟶ average ⟶ weak ⟶ ANY
	3	{3.0-3.49} ⟶ good ⟶ strong ⟶ ANY
	4	{3.5-4.0} ⟶ excellent ⟶ strong ⟶ ANY

Figure 3 Concept tree table and path id for each attribute

Name	Status	Sex	Age	GPA
John	freshman	M	20	3.2
Hack	freshman	M	19	2.8
Joe	junior	M	21	2.7
Mary	senior	F	22	3.3
Donald	M.A.	M	23	3.3
Loy	Ph.D.	M	26	3.2
Calvin	M.S.	M	26	3.6

Table 1 Initial relation of the university student database

Status path id	Sex path id	Age path id	GPA path id
1	1	1	3
1	1	1	2
3	1	1	2
4	2	1	3
5	1	1	3
7	1	2	3
6	1	2	4

Table 2 Path relation transformed from the initial relation

Once we have the Path relation, we can perform generalization on this relation. The user or the system would specify the generalization order of attributes of the relation. The concept trees of Sex, Age and GPA are shown in Figure 4 below. The initial relation is at levels 2,1,2,3 with respect to the attributes Status, Sex, Age and GPA. We can use vectors to represent the order of generalization. As in the student database, let $v_0 = \{2,1,2,3\}$ be the levels of corresponding attributes of the initial relation. If the user wants to generalize the attribute Age to level 1 first, then $v_1 = \{2,1,1,3\}$. Note that with this sequence of vectors, elements of v_i will be less than or equal to the corresponding element of v_j whenever $i \ge j$. Eventually, there is an integer t, where t is less than or equal to the sum of elements in v_0, such that $v_t = \{0,0,0,0\}$, this means that if the generalization is done t times, then all the tuples will be generalized to one tuple in the final relation. There is no restriction that each generalization step should generalize one attribute and one level only. The user can set one generalization step to generalize two or more attributes simultaneously and generalize any attribute by more than one level. For example, v_1 can be $\{2,1,1,2\}$,$\{2,1,2,1\}$ or even $\{2,1,1,1\}$.

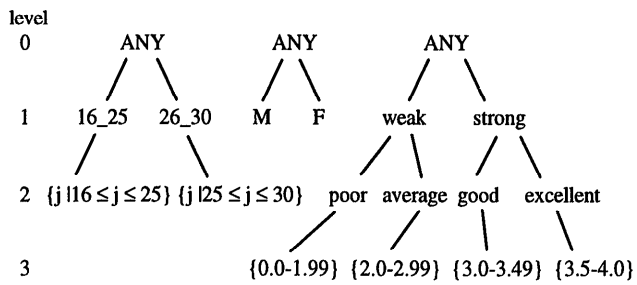


Figure 4 Concept trees and levels for the Sex, Age, GPA

Let us call the vectors as described above the **level vectors**. These vectors may be set by the user or generated by the system and can be set dynamically or statically. A dynamic approach would consider the attribute thresholds.

Let $A_1,...,A_n$ be the attributes not removed, suppose

$v_0 = \{e_{01},...,e_{0n}\}$,

$v_1 = \{e_{11},...,e_{1n}\}$,

...

$v_t = \{e_{t1},...,e_{tn}\} = \{0,0,...,0\}$

are the level vectors where e_{ij} is the level of concept of attribute j after i generalization steps. For each e_{ij}, there is a corresponding q_{ij} which is the number of distinct concepts in the concept tree at that level for attribute j.

The maximum possible size of the relation after i generalization steps, is given by q_i, where

$$q_i = \prod_{j=1}^{n} q_{ij} \qquad (1)$$

Let S_i be the set of tuples in the intermediate or final generalized relation after i generalization steps. Let $|S_i|$ be the number of tuples in S_i, we have $|S_i| \leq q_i$.

5. Path id generalization algorithm

Assume that the generalization relation threshold g is given, our task is to find the smallest s such that the number of tuples in the generalized relation, with attribute levels v_s, is smaller than or equal to g. We describe two approaches here.

Bottom-up approach

The system stores the information of Figure 4 and then use the Path relation to do generalization. At the i-th generalization step, a multi-dimensional array of integer VoteArray[1..q_{i1}][1..q_{i2}]...[1..q_{in}] of size q_i can be used. Each element in VoteArray corresponds to a unique combination of concepts at v_i. Alternatively, since the value of q_i may be large, allocating an array of this size may be inefficient or even impossible, in that case we can use a list VoteList to store the same information instead and insert the list by the B-tree method, where the index of the B-tree is the concept combination. As we shall see, the size of the list will not exceed the threshold g.

We describe in more details here the use of VoteArray. For example, with the previous student database, if v_1 = {1,1,1,2}, then we know that Status, Sex, Age, and GPA should generalize to level 1,1,1, and 2 respectively and they shall have 2,2,2,4 concepts respectively at the corresponding levels.

Status :	C_{11} = undergraduate	C_{12} = graduate	$q_{11} = 2$
Sex :	C_{21} = M	C_{22} = F	$q_{12} = 2$
Age :	C_{31} = 16_25	C_{32} = 26_30	$q_{13} = 2$
GPA :	C_{41} = poor	C_{42} = average	$q_{14} = 4$
	C_{43} = good	C_{44} = excellent	

The product of q_{11}, q_{12}, q_{13}, q_{14} is 2x2x2x4 = 32. We may use a multi-dimensional array VoteArray[1..2] [1..2][1..2][1..4], of size 32, to store the information during generalization.

Initially, each entry of VoteArray and a counter U is set to zero. Then the tuples of the Path relation is processed one by one. For each tuple T of the Path relation, each attribute's path id is generalized to a concept of that attribute. For example, with the tuple

Status path id	Sex path id	Age path id	GPA path id
1	1	1	3

we can find the corresponding concept of each path id from the path id table, e.g. we find that Status path id = 1 corresponds to the concept undergraduate, which is C_{11}. The other path id's corresponding concepts are C_{21}, C_{31}, C_{43}, so the concept combination is C_{11}, C_{21}, C_{31}, C_{43}. We shall increment one entry of VoteArray, that correspond to this concept combination, by 1. For example, the array index for the above concept combination is (1,1,1,3). If the value of this entry is equal to zero before the increment, we should increment the counter U by 1 and check if the counter U is greater than the threshold value g, if U > g then we should retrieve the next level vector and start the next generalization and reset the counter U to 0. If the counter is not greater than g, we then increment that entry of VoteArray and process the next tuple.

The remaining tuples of the Path relation are processed and VoteArray is incremented accordingly. After all the tuples of the Path relation is processed and if the counter U is still not greater than the threshold value g, we can form the final relation S. For each entry of VoteArray not equal to zero, we can find the tuple of concepts that the entry represents.

For the method using VoteList, we can keep a similar counter U, which counts the number of elements in the list. Similarly, the value of U is bounded by g and hence the size of VoteList is also bounded by g.

In fact, we can cut down on the computational complexity of the above: we notice that when we need to start the next generalization (as the counter is greater than g), we don't have to process the tuples in the Path relation which have been processed in previous generalizations again. This is because VoteArray or VoteList has stored the concept combinations and votes which these tuples generalize to. We can propagate the value of VoteArray or VoteList to the next generalization by using the concept tree table and then process the remaining tuples of the Path relation which have not been processed before. Hence, each tuple of the Path relation will be processed only once in the entire process in order to get the final relation.

Top-down approach

In viewing that the threshold g is small in general, a top down approach can be used instead of the bottom-up approach. Let v_t = {0,0,...,0}, we calculate q_t, q_{t-1}, ... by equation (1) to find the value w such that $q_w \leq g$ and $q_{w-1} > g$. Then we begin the process by retrieving v_w and forming VoteArray or VoteList accordingly. It is clear that $|S_w| \leq g$, but table S_w may not be the final relation we want since this may be an over-generalized table. We must retrieve the level vector v_{w-1} and check if the counter U_{w-1} is greater than g, if this is the case then S_w will be the final relation table we want. Otherwise, we retrieve the level v_{w-2} and check U_{w-2} accordingly, we repeat this procedure until there is an r, where $1 \leq r \leq t-1$, such that $U_r \leq g$ and the counter $U_{r-1} > g$, then S_r will be final relation we want.

Note that with the top down approach, we must generalize from the Path relation to the desired level each time. However, the top-down approach may need less number of generalization steps than the bottom-up approach.

6. Complexity of the algorithm

For the proposed method, we need O(N) time complexity to transform the initial relation to the Path relation. For the bottom-up approach, suppose we use a list to store the votes of the concept combinations at each generalization. (We can choose to use list at any generalization steps since the size of the VoteList is limited by the threshold g and the height of the B-tree is limited by log g. The time complexity of the i-th generalization will then be O(log g·N), which indicates the complexity of processing the tuples of Path relation and inserting them into the list by B-tree method. In the worst case, we have to do the generalization for t times, where t is the number of level vectors. Therefore, the time complexity of the algorithm is O(t·log g·N). We shall use the VoteArray method only if it can give better or similar performance. As stated at the end of the subsection on the Bottom-up approach, we can cut down the computational complexity of this algorithm. The complexity will be bounded by O(N·log g + t·g log g). Given that t and g are small and independent of N, the complexity is equal to O(N). Generally, the generalization relation threshold g is a small value, e.g. around 50, we can expect that the VoteList can be stored entirely in the main memory of the computer system, therefore, the number of disk I/O is $\frac{N}{B}$, where B is the number of tuples in a disk page. This is an improvement in comparison to the algorithm of DBLEARN [Han, Cai & Cercone 1993], which requires O($\frac{N \log N}{B}$·t) disk I/O as it has to merge tuples in the relation in each generalization step.

7. Generalization for Rule-based attribute-oriented approach

A rule-based hierarchy for background knowledge representation called Rule-Based Concept Graph is proposed in [Cheung, Fu & Han 1994]. In a rule-based concept graph, a concept can be generalized to more than one higher level concept, and rules are used to determine which generalization path should be taken. For example, with the university student database defined above, if the expectation for the graduate student is higher, we may have the set of conditional generalization rules for GPA as in Figure 5.

R_1 : {0.0-1.99} ⟶ poor
R_2 : {2.0-2.49} and {Status=graduate} ⟶ poor
R_3 : {2.0-2.49} and {Status=undergraduate} ⟶ average
R_4 : {2.5-2.99} ⟶ average
R_5 : {3.0-3.49} ⟶ good
R_6 : {3.5-3.79} and {Status=graduate} ⟶ good
R_7 : {3.5-3.79} and {Status=undergraduate} ⟶ excellent
R_8 : {3.8-4.0} ⟶ excellent
R_9 : {poor} ⟶ weak
R_{10}: {average} and {Status=senior or Status=graduate} ⟶ weak
R_{11}: {average} and {Status=freshman or Status=sophomore or Status=junior} ⟶ strong
R_{12}: {good} ⟶ strong
R_{13}: {excellent} ⟶ strong

Figure 5 Conditional generalization rules for GPA

For example, in R_2 of Figure 5, {Status=graduate} is a condition for generalizing the concept of GPA from {2.0-2.49} to poor.

Note that for the rule-based induction approach, there may be more than one path from a leaf to the root ANY in a concept hierarchy. We have to label each distinct path by a unique path id for each attribute. For example, the paths and their corresponding path ids of GPA may be labelled as follows, (the paths and path ids of other attributes are the same as before in Figure 3).

path 1: {0.0-1.99} ⟶ poor ⟶ weak ⟶ ANY
path 2: {2.0-2.49} ⟶ poor ⟶ weak ⟶ ANY
path 3: {2.0-2.49} ⟶ average ⟶ weak ⟶ ANY
path 4: {2.0-2.49} ⟶ average ⟶ strong ⟶ ANY
path 5: {2.5-2.99} ⟶ average ⟶ weak ⟶ ANY
path 6: {2.5-2.99} ⟶ average ⟶ strong ⟶ ANY
path 7: {3.0-3.49} ⟶ good ⟶ strong ⟶ ANY
path 8: {3.5-3.79} ⟶ good ⟶ strong ⟶ ANY
path 9: {3.5-3.79} ⟶ excellent ⟶ strong ⟶ ANY
path 10: {3.8-4.0} ⟶ excellent ⟶ strong ⟶ ANY

Once we have labelled the path id of each path in each attribute, we should reformulate the rules by replacing each concept C by a **path id list**. For example, suppose path 5, path 6, and path 7 of Status contain the concept graduate (see Figure 3), rule R_6 will be transformed to

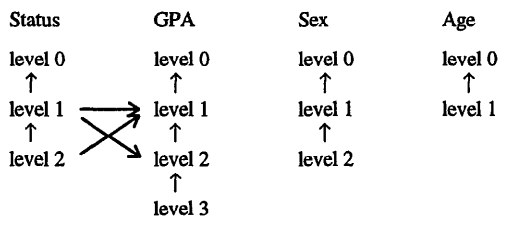
R_6' :{3.5-3.79} and {path_id(Status) ∈ {5,6,7}} ⟶ good

We call the list {5,6,7} a path id list of the concept graduate. The transformed rule will help us to determine the path id of each attribute in each tuple.

From R_2 in Figure 5, we see that the generalization on the attribute GPA depends on the value of attribute Status. We say that GPA **depends on** Status. Problems may arise if there are cycles of dependency. We make the assumption that with the given set of rules, if attribute A_1 depends on attribute A_2, attribute A_2 will not depend on A_1, and there does not exists a sequence of attributes {B_1, ..., B_r} such that A_1 depends on B_1, B_1 depends on B_2, ..., B_{r-1} depends on B_r, and B_r depends on A_1. This property ensure that no cycle of dependency can occur.

We can build a causal relation diagram for the generalization according to the rules. For the set of rules

in Figure 3 together with other unconditional generalization rules such as {M,F} ⟶ ANY(Sex), we could build the following causal relation diagram



Form the causal relation diagram, the generalization in attribute GPA depends on the condition of the attribute Status as there are arrows from Status to GPA. With this casual diagram, and by the assumption that there is no cycle of dependency, we can find at least one **dependency ordering** of attributes [$A_{p(1)}$, $A_{p(2)}$, ..., $A_{p(n)}$] which satisfies the following property,

(P1) If generalization of A_i depends on A_j, A_j will appear in the order list before A_i .

From the above causal relation diagram, the list [Sex, Age, Status, GPA] is one of the ordering of attributes which satisfy the above property.

After we get an ordering of attributes, we can build the Path relation for the entire database as in section 4. We can use the information of path id list in the generalization rules of each concept to transform the initial relation to the Path relation one attribute by one attribute according to the dependency ordering.

After forming the Path relation, we can perform generalization by the algorithm proposed in section 5 to get the final relation. However, for each generalization step, we have to process all tuples of the Path relation. Note that the Path relation in-corporated the information of the concept hierarchy. This eliminates the need of a "backtracking" procedure as described in [Cheung, Fu & Han 1994] and enhances performance. The complexity of this algorithm is O(t log g·N). Given that t and g are small and independent of N, the complexity is equal to O(N).

8. Conclusion

An important application of knowledge discovery is to support co-operative query answering [Motro & Yuan 1990]. With the attribute-oriented induction method, it is possible to ask query about high level concepts such as "what type of undergraduate students have strong GPA". This kind of query cannot be answered directly by querying the underlying database since the system does not understand the high level concepts like undergraduate or strong GPA.

In [Cai, Cercone & Han 1991, Han, Cai & Cercone 1992, Han, Cai & Cercone 1993], an attribute-oriented concept tree ascension technique has been proposed. The system DBLEARN applies this technique for knowledge discovering in large database. The performance of the system is good and the time complexity of the algorithm is O(N log N), where N is the number of tuples in the initial relation [Han, Cai & Cercone 1993]. As we find that the structure of the database and the set of generalization rules is static compare to the dynamic change of the data, preprocessing work can be done. In the preprocessing, the path id table is formed. Then we apply an efficient generalization process. We show that given the generalization threshold and concept hierarchies are small, the time complexity of our algorithm is O(N). The amount of disk I/O is O(log N) times less than the original method.

A Rule-based attribute-oriented approach, which is a generalized version of the attribute-oriented approach, has been proposed in [Cheung, Fu & Han 1994]. This rule-based attribute-oriented induction method can handle induction on a rule-based concept hierarchy. We have shown that the idea of path id is used to derive an efficient generalization algorithm. The costly backtracking in the original algorithm is eliminated and similar improvement in performance is achieved.

References

[1] Agrawal, R. 1994. Tutorial: Data Mining. In Proc. 13th ACM Symp. on Principles of Database Systems.

[2] Agrawal, R.; and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In Proc. 20th Intl Conf., VLDB.

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

[4] Cheung, D.; Fu, A.; and Han, J. 1994. Knowledge discovery in databases: A rule-based attribute-oriented approach. In 8 th Intl. Symp. of Methodologies for Intelligent Systems.

[5] Faloutsos, C.; and Lin, K. 1995. FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization. ACM SIGMOD Intl. Conf. on Management of Data.

[6] Frawley, W. J.; Piatetsky-Shapiro, G.; and Matheus, C. J. 1991. 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.

[7] Han, J.; Cai, Y.; and Cercone, N. 1992. Knowledge discovery in databases: An attribute-oriented approach. In Proc. 18th Intl. Conf., VLDB, pages 547-559, Canada.

[8] Han, J.; Cai, Y.; Cercone, N.; and Y. Huang. 1992. DBLEARN: A knowledge discovery system for databases. In Proc. 1st Intl. Conf. on Information and Knowledge Management, pages 473-481, Baltimore, Maryland, Nov.

[9] Han, J.; Cai, Y.; and Cercone, N. 1993. Data-driven discovery of quantitative rules in relational databases. IEEE Trans. Knowledge and Data Engineering, 5:29-40.

[10] Motro, A.; and Yuan, Q. 1990. Querying database knowledge. In Proceedings of 1990 ACM-SIGMOD Intl. Conf. on Management of Data, pages 173-183, Atlantic City, NJ.

[11] Park, J.; Chen, M.; and Yu, P. 1995. An Effective Hash Based Algorithm for Mining Association Rules. ACM SIGMOD Intl. Conf. on Management of Data.

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