Extending Attribute-Oriented Induction as a Key-Preserving Data Mining Method

Maybin K. Muyeba[1] and John A. Keane[1]

[1]Department of Computation, UMIST
P O BOX 88, Manchester, UK
muyeba, jak@co.umist.ac.uk

Abstract. Attribute-Oriented Induction (AOI) is a set-oriented data mining technique used to discover descriptive patterns in large databases. The classical AOI method drops attributes that possess a large number of distinct values or have either no concept hierarchies, which includes keys to relational tables. This implies that the final rule (s) produced have no direct link to the tuples that form them. Therefore the discovered knowledge cannot be used to efficiently query specific data pertaining to this knowledge in a different relation to the learning relation.

This paper presents the key-preserving AOI algorithm (AOI-KP) with two implementation approaches. The order complexity of the algorithm is *O (np)*, which is the same as for the enhanced AOI algorithm where *n* and *p* are the number of input and generalised tuples respectively. An application of the method is illustrated and prototype tool support and initial results are outlined with possible improvements.

1 Introduction

Data mining is the extraction of interesting patterns concealed in large databases [1]. There are various algorithms for extracting these patterns such as association, sequencing and classification [2]. *Attribute-Oriented induction* (AOI) [3] is a set-oriented generalisation technique used to find various types of rules.

The method integrates learning-from-examples techniques with database procedures. The AOI method basically involves three primitives that specify the learning task. These are collection of initial task-relevant data (*Data Collection*), use of background knowledge (*Domain knowledge*) [9] during the mining process and representation of the learning result (*Rule formation*). The fundamental principle in AOI is to generalise the *initial relation* to a *prime relation* and then to a *final relation* using background knowledge and user-defined threshold (s). Tuples found similar are merged as one with counts accumulated.

The AOI method is widely applicable and this underlines its importance. For example, mining characteristic and classification rules [3], multiple level generalisation [9] and cooperative or intelligent query answering [8]10].

J.M. Zytkow and J. Rauch (Eds.): PKDD'99, LNAI 1704, pp. 448-455, 1999.

The contribution of this paper is two-fold. Firstly, AOI is extended by preserving keys from the initial task relevant relation and associating them to the final rules while at the same time keeping the complexity of the key-preserving algorithm (AOI-KP) to *O(np)*, which is the same as for the enhanced AOI algorithm [4].

Secondly, the approach allows a user to make more efficient data queries on large data relations other than the learning relation if the preserved keys index this data. We are unaware of other specific work on AOI using the key-preserving method for performing such queries.

This paper is organised as follows: in section 2, related work is considered; in section 3 the proposed algorithm, AOI-KP, is presented and its complexity is discussed in section 4; in section 5 the application of the approach is illustrated; section 6 describes results from a prototype support tool for the method with further improvements and section 7 conclusion.

2 Related Work

The work builds on the classical AOI method introduced in [3]-[6], the rule-based approach [11] and intelligent query answering [10].

In [11], the information loss problem due to generalisation in the rule-based AOI is addressed by introducing a backtracking method using a generalised tuple called a *covering tuple.* However, the method cannot uniquely distinguish each covered tuple and so may be inefficient in performing queries directly on the discovered knowledge.

In [7], a framework is proposed for answering both data and knowledge queries to a knowledge base of substantial volume. Our work mainly concerns data queries on the usual database data [11].

In [8], key-preserving and key altering generalisations are discussed. If a key is generalised, information is lost if joins are made on the generalised relations.

Two memory-based AOI algorithms are reported in [3][4] and run in *O(n log n)* and *O(np)* times respectively, where *n* and *p* are the number of tuples in the input and generalised relations respectively. Further, two extensions to AOI that run in *O(n)* have been implemented [5,6]. However, in all these approaches, key preservation of the attribute identifying each tuple is not addressed to our requirements.

3 The Key-Preserving AOI Method

When data is retrieved, the system stores the table in memory and applies the induction process within the user-supplied thresholds. An implementation concern is how to determine the size of the key list a priori. The default value assumed could be the number of input tuples in the initial relation. The actual size can only be determined dynamically.

We introduce definitions relevant to the key-preserving algorithm. We assume an input relation with *n* tuples and each tuple is uniquely identified by a *key,* which is an integer value. Keys are stored in a static or dynamic array, here called a key list for convenience. We also assume that the keys are unaffected by database updates. For

any attribute A_i, (i = 1,..,n), let t_p and t_q be any two tuples with keys *p* and *q* such that $t_p(A_1) = p$, $t_q(A_1) = q$, $p<q$.

Definition 1. Equivalence. Two tuples t_p and t_q are said to be equivalent if $t_p(A_i) = t_q(A_i)$, $i \neq 1$, i=2,..,n.

Definition 2. Indexing key. A key *p* of tuple t_p is an *indexing key* in a row of a given key list if *p* is positioned in the *first* column of that row.

Definition 3. Ordinary key. A key *q* of tuple t_q is an *ordinary key* in a row of a given key list if *q* is positioned in *any* column of that row.

The AOI-KP algorithm is shown in figure 1. The first column of the key list is automatically sorted by the insertion method provided the initial input table is sorted. In the next section, we present the complexity of the algorithm and compare it with other algorithms developed in the literature.

4 Algorithm Complexity

AOI can be applied to large databases as it progressively reduces the search space (see figure 3). The initial problem is that all the examples forming the initial relation will be loaded into memory. Data mining algorithms should be both space and time efficient [5]. To save time, the construction of attribute concept hierarchies is done dynamically as the input is read to avoid re-scanning the input data or retrieving from disk. The algorithm proposed here works in the same way as AOI [4] but preserves the keys of the initial table.

The order complexity of AOI-KP is calculated as follows, assuming $p = n*k$, $0<k<1$, for large values of *n* where *p* is the number of tuples in the prime relation.

For a key list using static arrays with row and column size of up to *p* and *n* respectively, it is only necessary to find an empty column in the row of an equivalent tuple key. This means the row a key is positioned in the prime table is the same row it will be inserted in the key list. Thus, the worst time complexity for the key list insertion is $O(np)$ for the row linear space search. This gives a total order complexity of $O(np)$ plus $O(np)$ or $2*O(np)$, taking tuple insertion into account. In addition, the space required is $O(n+p)$ plus $O(np)$ for the data tables and the key list array respectively. This approach is, therefore, not space and time efficient[1].

However, when using dynamic arrays, the $O(np)$ linear search is eliminated as keys are only added to the dynamic array corresponding to the row of the prime table. The space required by this approach using dynamic arrays is $O(n+p)$ for both the initial and prime table. For *m* attributes where *m* is large (>10), the key list space requirement for dynamic arrays at any point during program execution is $O(n/m+c)$, where *c* is a small space increment due to equivalent tuples' key insertions which is much less than $O(np)$ for the static key approach.

[1] See Figure 4 of section 6

```
STEP 1: Collect Data and determine distinct values of Ai
BEGIN
  Declare Key_list array with two dimensions
  Assign first key to Key_list as an indexing key and insert the table
  FOR EACH attribute Ai  (1 ≤ i ≤n, Ai != A1) DO
   Construct hierarchy for attribute AI
   WHILE attribute threshold not reached DO
   BEGIN
     IF Ai  has no hierarchy THEN  Remove attribute Ai
       ELSE Substitute Ai by its next level generalised concept;
STEP 2: Merge any identical tuples tp, tq propagate counts, keys
         IF any two tuples tp,tq are equivalent THEN
         BEGIN
               Assign key p of tuple tp to variable KEY
           Look for indexing key KEY in Key_list to determine its row
           IF KEY is found THEN
           BEGIN
              Insert key (s) of tuple tq as ordinary key(s) in KEY's row
              Increase tuple count of inserted row in table
              Merge the tuples
            END
            ELSE BEGIN
                   Insert indexing key KEY, ordinary key q in next available row
                   Increase tuple count of inserted row in table
                   Merge the tuples
             END // end if key = found
          END
          ELSE IF next row in table is empty THEN
          BEGIN
              Insert tuple with key q in table
              Insert key q as indexing key in next empty row of Key_list
              Assign one to tuple count of inserted row in table
           END
       END// end if any two tuples
   END // end while
Repeat step 2 until entire table and key-list are checked.
STEP 3: Check tuple threshold and produce final table
   WHILE number of tuples is more than rule threshold DO
   BEGIN
         Selectively generalise an attribute with distinct value > rule threshold
        Merge tuples, propagate keys and increase counts using STEP 2.
END
 END. // Key-preserving AOI
```

Fig. 1. The Key-Preserving Algorithm (AOI-KP)

This gives a total order complexity of *O(np)* for large values of *n.* The space requirement is *O(n+p)* plus *O(n/m+c)* or simply *O(n)*. The dynamic key propagation approach therefore executes much faster and uses less memory than the static approach[2].

5 Application of the Approach

Consider a University database with the following schema:

Student (Sno, Sname, Sex, Major, Department, Birth_place, Residence)
Course (Cunit, Ctitle, Department, Ts, Te)
Exam (Cunit, Sno, GPA)
Calendar (Period, Acc_year, Ts, Te)
Registration (Sno, Rstatus, Ctitle, Acc_year)

where *Cunit* is course unit, *Ctitle* is course title, *Acc_year* is a user-defined time for academic year, *Sno* is student number, *Rstatus* is registration status and *Ts* and *Te* are time start and time end of the respective periods of study and courses.

In addition, assume the instance-based concept hierarchy for the attributes *Sex, Major* and *Birth_place* in figure 2. The initial table is retrieved with attributes *Sno, Sex* and *Birth_place* for learning about postgraduate students in a chosen academic year. the temporal attribute *'Acc_year'* is used in the WHERE clause of SQL thus making the discovered rules temporal.

{M.Sc, Ph.D,..}⇒postgraduate,
{Postgraduate, Undergraduate} ⊂ANY (Major)
{Female, Male} ⊂ANY (Sex)
{Bradford, Liverpool, Manchester, Edinburgh,.., London} ⊂ UK
{Chicago, Boston, New York,..,Washington} ⊂USA
{Calcutta, Bombay,.., New Delhi} ⊂India
{Shanghai, Nanjing,.., Beijing} ⊂China, {India, China,..} ⊂Asia
{UK, France, Germany,..} ⊂Europe
{USA, Canada,..} ⊂America, {America, Asia, Europe,..} ⊂ANY (Birth_place)

Fig. 2. Concept hierarchies for Sex, Major and Birth_place

After choosing a threshold of 3, the generalisation process preserves the key column *'Sno'* associated with each tuple. The generalisation process continues until thresholds are reached.

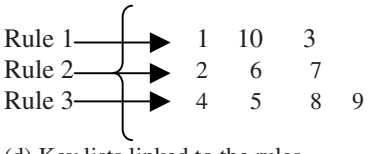
[2] See execution times for both methods on Figure 4 of section 6

Sno	Sex	Birth_place
1	M	USA
2	F	UK
3	F	USA
4	M	INDIA
5	M	INDIA
6	F	UK
7	F	UK
8	M	CHINA
9	M	CHINA
10	M	USA

Table 1. (a) Initial Table

Sno	Sex	Birth_place	Count
1	M	USA	2
2	F	UK	3
3	F	USA	1
4	M	INDIA	2
8	M	CHINA	2

Table 2. (b) Prime Table



(d) Key lists linked to the rules

Sno	Sex	Birth_place	Count
1	ANY	AMERICA	3
2	F	EUROPE	3
4	M	ASIA	4

Table 3. (c) Final Table

Fig. 3. Key propagation in the AOI process

The discovered rule would then be expressed in logic form as:

$\forall(x)\ Postgraduate(x) \wedge Acc\_year(x) = \text{“}Year\ 1\text{“} \Rightarrow$
$Birth\_place\ (x) \in America\ [30\%][Rule\ keys = 1, 3, 10]$
$\vee\ Sex(x) = \text{„}Female\text{“} \wedge Birth\_place\ (x) \in Europe\ [30\%][Rule\ keys = 2, 6, 7]$
$\vee\ Sex(x) = \text{„}Male\text{“} \wedge Birth\_place(x) \in Asia\ [40\%][Rule\ keys = 4, 5, 8, 9]$

Knowledge discovered in the AOI algorithm could be used to provide answers to data and knowledge queries [7,10]. For example, the rule *„All postgraduate students lived in private accommodation in year one* “ would be necessary to answer the queries: *„Did any postgraduate students live in University accommodation then?“ „Who were they?“ Which of them had an excellent GPA?*“ The first query can be answered from

the rule base. The latter queries, which involve listing and aggregation, can be answered efficiently with rule keys by querying the relevant tables. Moreover, the final generalised table, termed a *knowledge table*, can be used to join other data tables with *at least* the remaining tuples.

6. Prototype and Results

A prototype generalisation tool, *TAGET* (Temporal Attribute-oriented GEneralisation Tool) to mine temporal characteristic rules using the AOI approach is under development and initial results are shown in Figure 4. TAGET is a memory-based approach that uses data structures to store and manipulate data and concept

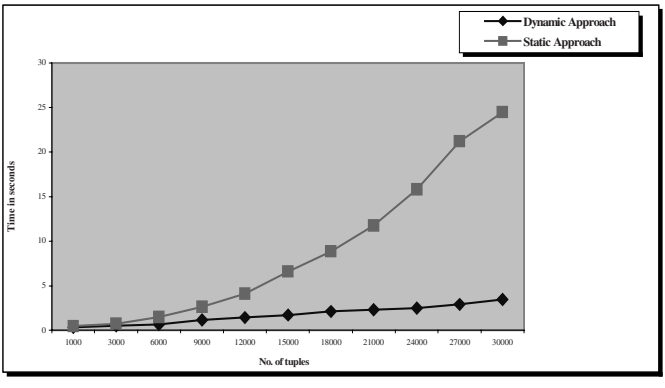


Fig. 4. TAGET Performance measures

hierarchies. Performance measures contrasting the dynamic and the static approaches for AOI-KP are shown in Figure 4 using an Intel P267 MHz with 64MB of memory and an attribute threshold of 3 as shown in Figure 3. Each set of input tuples was run five times and the average time computed. The initial results show that even with key propagation, the performance of AOI-KP is not drastically affected by the dynamic approach except memory limitations. We briefly explain how to handle this problem in the conclusion.

7. Conclusion

This paper has described an extension to AOI with key preservation (AOI-KP) that maintains the order complexity *O(np)* of AOG [4]. It has been shown using the

preserved Keys, the database of the discovered knowledge can be further interrogated more efficiently. For large databases, the output from the data queries that involve listing would be enormous for a single rule. This would need a threshold value on either the size of the key list or the number of keys required by the user.

TAGET can be further improved by using concurrency mechanisms during tuple and key insertion as well as file I/O for keys. In addition, assuming each tuple read will be generalised *at least once*, the order complexity can be improved to *O(n)* [5] by pre-generalisation. Using this method, the number of generalised tuples in memory will be orders smaller than the original task relevant data.

References

1. Frawley, W. J. and Piatetsky-Shapiro, G. 1991. „*Knowledge Discovery in Databases*“, AAAI/MIT Press, pp 1-27.
2. Agrawal, R.; Imielinski, T. and Swami, A. 1993. „*Database Mining: A performance perspective*“ IEEE Transactions on Knowledge and Data Engineering, 5(6):914-925, December.
3. Han, J; Cercone, N. and Cai, Y. 1991 „*Attribute-Oriented Induction in Relational Databases*“ In G. Piatetsky-Shapiro and W. J. Frawley, editors, Knowledge Discovery in Databases, pp 213-228.
4. Han, J.; Cai, Y. and Cercone, N. 1993. „*Data-Driven Discovery of Quantitative Rules in Relational Databases*“ IEEE Transactions on Knowledge and Data Engineering, 5(1):29-40, February.
5. Carter, C. L. and Hamilton, H. J. 1998. „*Efficient Attribute-Oriented Generalisation for Knowledge Discovery from Large Databases*“ IEEE Transactions on Knowledge Discovery and Data Engineering, 10(2):193-208, March.
6. Hwang, H; and Fu, W. 1995. „*Efficient algorithms for Attribute-Oriented Induction*“ In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD’95), pp 168-173, Montreal, Quebec.
7. Motro, A. and Yuan, Q. 1990. „*Querying Database Knowledge*“ In Proceedings of the ACM-SIGMOD International Conference on Management of Data, pp 173-183, Atlantic City, NJ.
8. Han, J.; Fu, Y. and Ng, R. T. 1994. „*Cooperative Query Answering Using Multiple Layered Databases*“ In Proceedings of the International Conference on Cooperative Information Systems (COOPIS ’94), pp 47-58, Toronto, Canada.
9. Fu, Y. 1996. „*Discovery of Multiple-Level rules from Large Databases*“ Ph.D. thesis, Computing Science, Simon Fraser University, July.
10. Han, J.; Huang, Y.; Cercone, N. and Fu, Y. 1996. *„Intelligent Query Answering by Knowledge Discovery Techniques*“ IEEE Transactions on Knowledge and Data Engineering, 8(3):373-390, June.
11. Cheung, D. W.; Fu, A. W.-C and Han. J. 1994. *„Knowledge Discovery in Databases: A Rule-Based Attribute-Oriented Approach*“ In Proceedings of the 1994 International Symposium on Methodologies for Intelligent Systems (ISMIS ’94), pp 164-173, Charlotte, North Carolina.