Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 166 (2005) 221–245
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Computing, Artificial Intelligence and Information Technology

Mining generalized knowledge from ordered data through attribute-oriented induction techniques

Yen-Liang Chen *, Ching-Cheng Shen

Department of Information Management, National Central University, Chung-Li, Taiwan 320, Republic of China

Received 2 December 2002; accepted 14 April 2004
Available online 25 June 2004

Abstract

The attribute-oriented induction (AOI for short) method is one of the most important data mining methods. The input of the AOI method contains a relational table and a concept tree (con\cept hierarchy) for each attribute, and the output is a small relation summarizing the general characteristics of the task-relevant data. Although AOI is very useful for inducing general characteristics, it has the limitation that it can only be applied to relational data, where there is no order among the data items. If the data are ordered, the existing AOI methods are unable to find the generalized knowledge. In view of this weakness, this paper proposes a dynamic programming algorithm, based on AOI techniques, to find generalized knowledge from an ordered list of data. By using the algorithm, we can discover a sequence of K generalized tuples describing the general characteristics of different segments of data along the list, where K is a parameter specified by users.

Keywords: Attribute-oriented induction; Concept hierarchy; Data mining; Relational data; Ordered data; Dynamic programming

1. Introduction

Data mining extracts implicit, previously unknown and potentially useful information from databases. Many approaches have been proposed to extract information. According to the classification scheme proposed in recent surveys (Chen et al., 1996; Han and Kamber, 2001), one of the most important ones is the attribute-oriented induction (AOI) method. This approach was first introduced in Cai et al. (1990), Han et al. (1992), Han et al. (1993).

* Corresponding author. Tel.: +886 3 4267266; fax: +886 3 4254604.
E-mail address: ylchen@mgt.ncu.edu.tw (Y.-L. Chen).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2004.04.029

The AOI method was developed for knowledge discovery in relational databases. The input of the method includes a relation table and a set of concept trees (concept hierarchies) associated with the attributes (columns) of the table. The table stores the task-relevant data, and the concept trees represent the background knowledge. The core of the AOI method is on-line data generalization, which is performed by first examining the data distribution for each attribute in the set of relevant data, calculating the corresponding abstraction level that the data in each attribute should be generalized to, and then replacing each data tuple with its corresponding generalized tuple. The major generalization techniques used in the process include attribute-removal, concept-tree climbing, attribute-threshold control, propagation of counts and other aggregate values, etc. Finally, the generalized data is expressed in the form of a generalized relation from which many kinds of rules can be discovered, such as characteristic rules and discrimination rules. For more details, please refer to the original papers (Cai et al., 1990; Han et al., 1992; Han et al., 1993).

Undoubtedly, the AOI method has achieved a great success. Because of its success, extensions have been proposed in the following directions: (1) extensions and applications based on the basic AOI approach (Han et al., 1993; Han et al., 1998; Lu et al., 1993), (2) more efficient methods of AOI (Carter and Hamilton, 1995; Carter and Hamilton, 1998; Cheung et al., 2000), (3) more general background knowledge (Hamilton et al., 1996; McClean et al., 2000), (4) integrating AOI with other information reduction methods (Hu and Cercone, 1996; Shan et al., 1995) and (5) proposing new variants of generalized rules (Tsumoto, 2000).

Existing research on AOI is based on set-oriented operations. Thus, current AOI-based methods can only find generalized knowledge from a set of relational data, where there is no order among the data. If the data are ordered, current methods are not able to find any ordered generalized knowledge. Unfortunately, ordered data frequently occurs in practice. Possible examples include a list of universities ordered according to their national rankings, a list of students arranged according to their GPAs, a list of mutual funds sorted in terms of their total returns and a list of salespersons sorted according to their sales figures. In all these examples, it will be very helpful for decision makers if we can identify the characteristics of different segments of data along the list. For example, if the data is concerned with salespeople, then we can identify what characteristics commonly occur in the best sellers, and what characteristics are commonly seen in the poor sellers. Such characteristics can be used not only to determine the criteria we should use when recruiting new employees but also the way we should educate and train the current sellers. When the data is about students, we can foresee what kinds of students will possibly have good GPAs' in the future, and therefore we can actively look for students who have this potential. When the data concern mutual funds for a year, we can describe which kinds of mutual funds are more competitive and profitable and which are not. With this information, the mutual fund company can adjust the investment portfolio, strategy, managers and analysts. As for the customers, they can use this information to choose promising investment funds.

To induce the characteristics of different segments of data along the list, say K segments, two steps are performed. First, we partition the data into K consecutive segments. Next, we find the characteristics of the data in each segment by using the generalization techniques developed by AOI. Of these two steps, the second one can be performed easily by generalizing the data in the same segment into a generalized tuple through concept-tree climbing. On the other hand, the first job is more difficult to do, because there are C_K^n possible ways we can partition the data, where n is the total number of data. Faced with so many possible choices, we require an efficient method of determining the best partition that produces K generalized ordered tuples. To this end, we first define a distance metric to measure the distance between generalized tuples, and then we propose a dynamic-programming algorithm to find the minimum distance partition as well as the corresponding K generalized ordered tuples.

In short, the contribution of this paper is twofold. First, AOI methods are extended to deal with ordered data, because preciously AOI methods could only deal with relational data, which seriously restricted their applicability. With the proposed method, we can discover a sequence of K generalized tuples describing the

characteristics of different segments of data along the list. Second, this paper gives a way to formally define the distance between generalized tuples. This opens a new direction for future AOI research.

This paper is organized as follows. In Section 2, we will use an example to explain the traditional AOI algorithm. In the same section, we also define our problem and explain the differences between our problem and the traditional AOI problem. In Section 3, the algorithm consisting of five phases is introduced, and its time and space complexities are given. Next, we deal with some potential problems that we may encounter when applying the proposed algorithm to real data sets. Four problems are identified. To remedy these four problems, three extensions of the algorithm are proposed in Section 4. The results of our performance evaluation are given in Section 5. Finally, the conclusions are given in Section 6.

2. The AOI algorithm and problem definition

AOI takes as input a database relation and a concept hierarchy for each attribute. A concept hierarchy is a tree structure that forms a taxonomy of concepts ranging from a single, most general concept at the root to some representation of all attribute values at the leaves. Higher level concepts are formed by grouping related values together and representing these by a single symbol. Concept trees may be defined for either discrete or continuous valued attributes. A leaf concept for a continuous attribute is expressed as a range of values. In Appendix A, four example concept trees are shown.

An AOI generalization task is initiated when a user specifies a relation to be retrieved and a set of concept trees to use in generalizing the data. First, the data values are converted to leaf concepts from the appropriate concept trees. The relation is then generalized by replacing leaf concepts with more general concepts. As concepts are generalized, the number of distinct concepts of each attribute is reduced, and many tuples become redundant as the information they contain becomes identical to other tuples. These are eliminated by removing all but one of the identical tuples. The number of original tuples associated with each remaining tuple is tracked in the tuple's count field. The generalization process is limited by two thresholds: the attribute threshold and the table threshold. The attribute threshold specifies the maximum number of distinct values of any attribute that may exist after generalization, and the table threshold gives an upper bound on the number of the generalized tuples that remain after the generalization process. When the input relation has been generalized to such an extent, generalization ceases. The result is a small set of tuples that displays the general characteristics of the input relation.

Throughout the paper, we use the sample table shown in Table 1 in our examples. It has 10 tuples and 4 attributes. In addition, we use $SI(i,r)$ to denote the value of attribute r of data tuple i. All concept hierarchies are shown in Appendix A. Suppose we set the attribute threshold as 2 and view the data in Table 1 as

Table 1
A sample table of 10 tuples and 4 attributes

$SI(i,r)$	Light Vehicle Model	Location of Manufacturer	Engine Displacement	Price
1	Compact SUV	Honda	2000	19,800
2	Midsize SUV	Mazda	2200	24,000
3	Fullsize Van	Chevrolet	4200	56,000
4	Compact Van	Chevrolet	3200	45,600
5	Compact Car	Volkswagen	1810	25,100
6	Midsize Car	Volkswagen	2300	30,500
7	Midsize Car	BMW	3000	32,000
8	Compact SUV	Nissan	2190	22,300
9	Compact SUV	Toyota	2000	19,800
10	Midsize SUV	Toyota	2799	24,500

Table 2
After generalizing the sample data with AOI

$SO(i,r)$	Light Vehicle Model	Location of Manufacturer	Engine Displacement	Price	Count
1	Light truck	Any	Middle	Low price	5
2	Light truck	Any	High	High price	2
3	Car	Any	Middle	High price	3

unordered. Then the result after applying the AOI method to Table 1 is Table 2, where $SO(i,r)$ denotes the generalized value of attribute r of tuple i. The first generalized tuple in Table 2 comes from tuples 1, 2, 8, 9 and 10 in Table 1, the second from tuples 3 and 4 and the third from tuples 5, 6 and 7.

Now, we describe our problem, the generalization from ordered data problem. The input parameters of our problem are similar to those of the AOI method, but we assume that the input table has been sorted according to some criteria specified by a decision maker. For example, suppose we are given the sales data of a company in the last five years ordered by date. We are required to generalize the data while keeping it ordered by date.

Besides the difference mentioned above, another difference is that instead of the attribute and table thresholds, we requires the number of partitions K and two parameters α and β for weighting (the purpose of these two parameters is explained in the next section). The knowledge we want to discover is the characteristics of different segments of the data along the list.

If we apply our algorithm to the table in Table 1 by viewing the table as ordered and setting the value of K as 4, then the result is as shown in Table 3. Suppose that the table in Table 1 was sorted by the ages of customers, from youngest to oldest. Table 3 indicates that older and younger customers have the similar purchasing behavior.

The first generalized tuple in Table 3 comes from tuples 1 and 2 in Table 1, the second from tuples 3 and 4 in Table 1, the third from tuples 5, 6, 7 and the last from tuples 8, 9 and 10. The traditional AOI method uses threshold values to control the generalization degree, but we use distance to control the degree of generalization. Because of this difference, the traditional AOI always presents a summary at a single level of granularity, while our method is capable of summarizing non-overlapping subsets of tuples to different levels of granularity. For example, the column "location of manufacturer" in Table 3 is generalized to two different levels, where the first, the third and the fourth generalized tuples in Table 3 are at a higher level than the second tuple.

As well, the two generalized tables must be interpreted differently. The table in Table 2 means that 50% of data in the target class have the characteristics of (Light truck, Any, Middle, Low price), 20% of data have (Light truck, Any, High, High price) and 30% of data (Car, Any, Middle, High price). The table in Table 3 indicates that the first segment of data in the list corresponding to the youngest customers has the characteristics of (SUV, Japan, 1601-2500, Economic), the second segment has (Van, Chevrolet, High, Expensive), the third segment has (General Car, Germany, Middle, 25000-34999) and the last segment has (SUV, Japan, Middle, Economic).

Table 3
The result after applying our algorithm

$SO(i,r)$	Light Vehicle Model	Location of Manufacturer	Engine Displacement	Price
1	SUV	Japan	1601-2500	Economic
2	Van	Chevrolet	High	Expensive
3	General Car	Germany	Middle	25000-34999
4	SUV	Japan	Middle	Economic

3. The Ordered AOI algorithm

To solve the stated problem, we propose the Ordered AOI algorithm. This algorithm partitions the data into a set of consecutive segments, from each of which the characteristics are found. Partitioning should not be done arbitrarily. Instead, we are looking for a partition satisfying two goals: (1) the data in the same segment should be as similar as possible; (2) the characteristics between different segments should be as different as possible. To meet these two goals, the problem becomes how to minimize the intra-segment distance inside the same segments but at the same time maximize the inter-segment distance between successive segments. Therefore, this paper defines two distance metrics to measure these two types of distances. We use a weighted summation of them to measure which partition is the best one.

The inputs of the Ordered AOI algorithm include three parts: (1) an ordered list of data tuples; (2) the concept tree for each attribute; and (3) the value of K. The output consists of K generalized tuples, where a generalized tuple is the generalization of those data tuples in the same segment.

Since the Ordered AOI algorithm contains five phases, Section 3.1 gives an overview of the entire algorithm. After that, we introduce theses five phases in order from Sections 3.2–3.6. Finally, Sections 3.7 and 3.8 give the time and space complexities of the algorithm, respectively.

3.1. An overview of the Ordered AOI algorithm

The first phase of the Ordered AOI algorithm computes the conceptual distances between every pair of consecutive tuples for every attribute. The values of attribute r in the ith tuple and the $(i+1)$th tuple will become the same if they climb high enough along the concept hierarchy for attribute r. Let us refer to the number of steps climbed as the generalization height of tuple i for attribute r, denoted as $F(i,r)$. Semantically, $F(i,r)$ denotes the distance of attribute r between tuples i and $i+1$. Our first phase computes $F(i,r)$ for all tuples i and all attributes r.

Based on $F(i,r)$, the second phase computes $E(i,j,r)$ for all i, j and r, which denotes the minimum number of steps we need to climb up along the concept hierarchy of attribute r before the data from tuple i to tuple j will have the same value. Semantically, $E(i,j,r)$ denotes the distance of attribute r for the data from tuple i to tuple j.

The third phase is to compute the intra-segment distance of a segment of tuples from tuple i to tuple j, denoted as $DI(i,j)$, for all i and j. $DI(i,j)$ can be obtained by computing the weighted summation of $E(i,j,r)$ for all attributes r, since the former denotes the distance for all attributes while the latter denotes the distance for a single attribute.

In the fourth phase, we have two major jobs. The first job is to compute the inter-segment distance between two adjacent segments of tuples, denoted as $DS(i,j,k)$, where the first segment is from tuple i to tuple j while the second is from tuple $j+1$ to tuple k. To do so, we first give the definition of $DS(i,j,k)$. After that, we consider how to compute it in an easier way, and we show that it can be computed directly from $DI(i,j)$, which was obtained in the preceding phase.

Based on $DI(i,j)$ and $DS(i,j,k)$, the second job of the fourth phase is to determine the best partition points for a given set of data tuples by using the dynamic programming method. One result obtained, called $D(i,j,s)$, tells us the distance of the optimum partitioning of the data from tuple i to the last tuple into s segments subject to the constraint that the first segment is from tuple i to tuple j. Complementarily, another result, called $B(i,j,s)$, tells us the first partition point after tuple j that achieves the optimum distance of $D(i,j,s)$. At the end of the fourth phase, we can find the optimum distance by choosing the optimum one from $D(1,j,K)$ for all j.

Finally, our last phase is to compute all the partition points required to achieve the optimum distance, denoted $D(1,j^*,K)$. These points are obtained by repetitively going backward from $B(1,j^*,K)$. After finding all partition points, it is easy to derive the generalized tuples, and we report them as the output knowledge.

3.2. The first phase

Recall that the first phase computes the conceptual distances between every pair of consecutive tuples for every attribute. Let $SI(i,r)$ denote the value of attribute r of data tuple i. A concept tree of height L_r is associated with attribute r, meaning that the levels of the tree are from level 0 to level L_r and the root node is at level 0. As with much previous research in AOI, we assume that all the leaves in a concept tree are at the same level. In the beginning, the value $SI(i,r)$ is at level L_r of the tree. Afterwards, it may climb up a certain number of steps along the tree, and we refer to the number of steps climbed as the generalization height. Let p_r denote the level of the node after $SI(i,r)$ has been generalized. Then, the generalization height of attribute$r$ of tuple i equals to $L_r - p_r$.

Any two successive values $SI(i,r)$ and $SI(i+1,r)$ can be generalized to the same value if they climb up along the tree to the nearest common ancestor. Suppose their nearest common ancestor is at level p_r. Then the generalization height of attribute r between tuple i and tuple $i+1$ equals to $L_r - p_r$, denoted as $F(i,r)$. It is easy to see that $F(i,r)$ can be determined in time $O(L_r)$ for given i and r, because we climb at most $O(L_r)$ steps in the tree. For n tuples and m attributes, the time is $O(m \times n \times L_r)$.

In the following, we give the algorithm Comp-F to compute $F(i,r)$ for all tuples and for all attributes.

Procedure: Comp-F

```
Begin
  For i ← 1 to n − 1 do //n, the number of tuples
   For r ← 1 to m do //m, the number of attributes
          Get p_r by finding the nearest common ancestor of SI(i,r) and SI(i + 1, r)
          F(i,r) ← L_r − p_r
   End
  End
End;
```

Example 1. The data table shown in Table 1 has four attributes, and their concept trees are shown in Appendix A. We have $L_1 = 3$, $L_2 = 3$, $L_3 = 2$ and $L_4 = 3$. By applying the above algorithm, we get the result shown in Table 4. Here, let us take a closer look at how the value of $F(4,2)$ is computed. This entry is determined by $SI(4,2)$ and $SI(5,2)$, which are Chevrolet and Volkswagen, respectively. By examining the concept tree of attribute 2, we find that their nearest common ancestor is Any, which is 3 levels higher than each of these two elementary values. Thus, we have the result that $F(4,2) = 3$.

3.3. The second phase

The goal of this phase is to compute the generalization height of attribute r for a segment of consecutive tuples from tuple i to tuple j, denoted as $E(i,j,r)$. Similar to the definition in the foregoing section, we define it as the number of steps that we must climb along the tree before all these tuples will have the same value. To this end, we must compute the nearest common ancestor of $SI(i,r)$, $SI(i+1,r),\ldots,SI(j,r)$ in the concept tree of attribute r. Let p_r be the level of this ancestor node in the tree. Then the generalization height $E(i,j,r)$ of attribute r of the data from tuple i to tuple j equals to $L_r - p_r$.

In order to simplify the computation, we use the following lemma to compute $E(i,j,r)$ from $F(k,r)$, which is already known from the preceding phase.

Lemma 1. $E(i,j,r) = \underset{i \leqslant k \leqslant j-1}{\text{maximum}}(F(k,r))$ *if* $i \leqslant j-1$ *and* $E(i,j,r) = 0$ *if* $i = j$.

Table 4
$F(i,r)$ for the data in Table 1

$F(i,r)$	1	2	3	4
1	1	1	0	1
2	2	3	2	3
3	1	0	1	1
4	3	3	2	2
5	1	0	0	0
6	0	1	1	0
7	3	3	1	3
8	0	1	0	1
9	1	0	1	1

Proof. We use induction to prove the lemma. The basis case is $i+1=j$. This case holds trivially because $E(i,j,r)=E(i,i+1,r)=F(i,r)$ by definition. Next, assume that the lemma is true for $E(i,k,r)$ where $k<j$. And we now consider the case of $E(i,j,r)$. Our proof is based on the observation that, if the nearest common ancestors are the same, then the generalization heights are also the same, and vise versa. So, let $NCA(i,j,r)$ denote the nearest common ancestor of the data from $SI(i,r)$ to $SI(j,r)$. Then, $NCA(i,j-1,r)$ must be a certain node along the path from node $SI(j-1,r)$ to the root node. Similarly, $NCA(j-1,j,r)$ is also a certain node on the same path. If $NCA(i,j-1,r)$ is at a level higher than that of $NCA(j-1,j,r)$, then $NCA(i,j-1,r)$ is an ancestor of $NCA(j-1,j,r)$. Otherwise, $NCA(j-1,j,r)$ becomes an ancestor of $NCA(i,j-1,r)$. No matter what case it is, the one in the higher level becomes $NCA(i,j,r)$, for this node is not only an ancestor of the data from $SI(i,r)$ to $SI(j-1,r)$ but also an ancestor of the data from $SI(j-1,r)$ to $SI(j,r)$. As a result, the generalization height of the data from $SI(i,r)$ to $SI(j,r)$ equals to the maximum of that from $SI(i,r)$ to $SI(j-1,r)$ and that from $SI(j-1,r)$ to $SI(j,r)$. By the induction basis and hypothesis, this implies that $E(i,j,r)$ is equal to the maximum of $F(j-1,r)$ and $\underset{i\leqslant k\leqslant j-2}{\text{maximum}}\{F(k,r)\}$.

Therefore, we have the conclusion that $E(i,j,r)=\underset{i\leqslant k\leqslant j-1}{\text{maximum}}(F(k,r))$. □

Example 2. For explanation, let us use the data in Table 1 to check how to compute $E(4,8,4)$, where we have $SI(4,4)=45{,}600$, $SI(5,4)=25{,}100$, $SI(6,4)=30{,}500$, $SI(7,4)=32{,}000$ and $SI(8,4)=22{,}300$. By examining the concept tree of attribute 4, we see that the nearest common ancestor of the data from $SI(4,4)$ to $SI(7,4)$ is node "High price". On the other hand, the nearest common ancestor of the data from $SI(7,4)$ to $SI(8,4)$ is node "Any price". Since node "Any price" is an ancestor of node "High price", the generalization height of the data from $SI(4,4)$ to $SI(8,4)$ is equal to that from $SI(7,4)$ to $SI(8,4)$. That means, $E(4,8,4)=\max E(4,7,4)$, $E(7,8,4)=E(7,8,4)$. According to the induction basis and hypothesis of Lemma 1, we have $E(7,8,4)=\max\{3\}=3$ and $E(4,7,4)=\max\{2,0,0\}=2$. Therefore, we have the result that $E(4,8,4)=\max\{2,0,0,3\}=3$.

Based on Lemma 1, an algorithm is given below to find all $E(i,j,r)$ for all pairs of tuple positions and attributes.

Procedure: Comp-E

Begin
For $r \leftarrow 1$ to m do //m, the number of attributes
For $i \leftarrow 1$ to $n-1$ do //n, the number of tuples

```
                E(i, i, r) ← 0
                E(i, i + 1, r) ← F(i, r)
                For j ← i + 1 to n − 1 do
                    If E(i,j,r) < F(j,r) Then
                        E(i, j + 1, r) ← F(j, r)
                    Else
                        E(i, j + 1, r) ← E(i, j, r)
                    End
            End
        End
    End;
```

Example 3. By applying the above algorithm to the data in Table 1 and the $F(i,r)$ table in Table 4, we get the result shown in Table 5.

3.4. The third phase

The third phase is to compute the intra-segment distance of a segment of tuples from tuple i to tuple j, denoted as $DI(i,j)$, for all i and j. Recall that $DI(i,j)$ can be obtained by computing the weighted summation of $E(i,j,r)$ for all attributes r, since the former denotes the distances for all attributes while the latter denotes

Table 5
$E(i,j,1)$ for the data in Table 1

$E(i,j,1)$	1	2	3	4	5	6	7	8	9	10	$E(i,j,2)$	1	2	3	4	5	6	7	8	9	10
1	0	1	2	2	3	3	3	3	3	3	1	0	1	3	3	3	3	3	3	3	3
2		0	2	2	3	3	3	3	3	3	2		0	3	3	3	3	3	3	3	3
3			0	1	3	3	3	3	3	3	3			0	0	3	3	3	3	3	3
4				0	3	3	3	3	3	3	4				0	3	3	3	3	3	3
5					0	1	1	3	3	3	5					0	0	1	3	3	3
6						0	0	3	3	3	6						0	1	3	3	3
7							0	3	3	3	7							0	3	3	3
8								0	0	1	8								0	1	1
9									0	1	9									0	0
10										0	10										0

$E(i,j,3)$	1	2	3	4	5	6	7	8	9	10	$E(i,j,4)$	1	2	3	4	5	6	7	8	9	10
1	0	0	2	2	2	2	2	2	2	2	1	0	1	3	3	3	3	3	3	3	3
2		0	2	2	2	2	2	2	2	2	2		0	3	3	3	3	3	3	3	3
3			0	1	2	2	2	2	2	2	3			0	1	2	2	2	3	3	3
4				0	2	2	2	2	2	2	4				0	2	2	2	3	3	3
5					0	0	1	1	1	1	5					0	0	0	3	3	3
6						0	1	1	1	1	6						0	0	3	3	3
7							0	1	1	1	7							0	3	3	3
8								0	0	1	8								0	1	1
9									0	1	9									0	1
10										0	10										0

the distance for a particular attribute. Formally, the computation can be expressed by the following equation:

$$DI(i,j) = \sum_{r=1}^{m} E(i,j,r) \times \chi_r,$$

where χ_r is a constant reflecting the relative importance of attribute r. Since this computation is easy, we omit the corresponding algorithm. However, if we apply the above equation to compute the intra-segment distances for the data in Table 1 and set all χ_r equal to 1, then we obtain the result shown in Table 6.

3.5. The fourth phase

In the fourth phase, we have two major jobs. The first job is to compute the inter-segment distance between two adjacent segments of tuples, and the second is to determine the best partition points for a given set of data tuples by using a dynamic programming approach.

Let the inter-segment distance between two adjacent segments of tuples be denoted as $DS(i,j,k)$, where the first segment is from tuple i to tuple j and the second segment is from tuple $j+1$ to tuple k. Assume that the data from tuple i to tuple j can be generalized into a generalized tuple P with m attribute values $P_1, P_2, \ldots, P_m$. For each value P_r, let LP_r be its level in the concept tree of attribute r. Similarly, assume that the data from tuple $j+1$ to tuple k can be generalized into a generalized tuple Q with attribute values $Q_1, Q_2, \ldots, Q_m$ and levels $LQ_1, LQ_2, \ldots, LQ_m$. Further, assume that T denotes the generalized tuple produced from tuple i to tuple k, and that its values are $T_1, T_2, \ldots, T_m$ and their levels are $LT_1, LT_2, \ldots, LT_m$.

Obviously, P and Q are adjacent generalized tuples, and T denotes the generalization of P and Q until a common ancestor is reached. The distance required to generalize P into T is $\sum_{r=1}^{m}(LP_r - LT_r) \times \chi_r$. Similarly, the distance required to generalize Q into T is $\sum_{r=1}^{m}(LQ_r - LT_r) \times \chi_r$. Summing both terms together, we obtain the total distance to generalize tuple P as well as tuple Q into a single tuple T. Therefore, $DS(i,j,k)$ can be written as $\sum_{r=1}^{m}(LP_r - LT_r) \times \chi_r + (LQ_r - LT_r) \times \chi_r$. Since the following lemma shows that $\sum_{r=1}^{m}(LP_r - LT_r) \times \chi_r = DI(i,k) - DI(i,j)$, the formula of $DS(i,j,k)$ can be simplified as

$$DS(i,j,k) = DI(i,k) - DI(i,j) + DI(i,k) - DI(j+1,k). \tag{1}$$

Table 6
$DI(i,j)$ for the data in Table 1

$DI(i,j)$	1	2	3	4	5	6	7	8	9	10
1	0	3	10	10	11	11	11	11	11	11
2		0	10	10	11	11	11	11	11	11
3			0	3	10	10	10	11	11	11
4				0	10	10	10	11	11	11
5					0	1	3	10	10	10
6						0	2	10	10	10
7							0	10	10	10
8								0	2	4
9									0	3
10										0

Lemma 2. $\sum_{r=1}^{m}(LP_r - LT_r) \times \chi_r = DI(i,k) - DI(i,j)$.

Proof. We use the following derivation for the proof.

$$\sum_{r=1}^{m}(LP_r - LT_r) \times \chi_r = \sum_{r=1}^{m}(L_r - LT_r) \times \chi_r - (L_r - LP_r) \times \chi_r = \sum_{r=1}^{m} E(i,k,r) \times \chi_r - E(i,j,r) \times \chi_r$$
$$= \sum_{r=1}^{m} E(i,k,r) \times \chi_r - \sum_{r=1}^{m} E(i,j,r) \times \chi_r = DI(i,k) - DI(i,j) \qquad \square$$

Based on Eq. (1), it is easy to compute the inter-segment distance between any two successive segments of data tuples. For conciseness, we omit the algorithm.

Suppose all n input tuples have been partitioned into K consecutive segments. Then we will have K intra-segment distances and $(K-1)$ inter-segment distances. Thus, we define the total distance as

$$\alpha \times (\text{the sum of all intra} - \text{segment distances}) - \beta \times (\text{the sum of all inter} - \text{segment distances}).$$

Here, we use a minus operator between these two distances because we wish the former as small as possible but the latter as large as possible. Our goal is then to find a partition of tuples such that the total distance will be minimized.

Let $D(i,j,s)$ denote the minimum total distance of the data from tuple i to the last tuple if we partition it into s segments and the first segment is from tuple i to tuple j. Complementarily, $B(i,j,s)$ denotes the first partition point after tuple j that achieves the optimum distance of $D(i,j,s)$. If $s=1$, we have the following base relation.

$$D(i,j,s) = \alpha \times DI(i,n), \quad \text{if } j = n,$$
$$D(i,j,s) = \text{undefined if } j \neq n,$$
$$B(i,j,s) = \text{undefined}.$$

In case that $s=2$ and $n>j$, the recursive relation becomes as follows:

$$D(i,j,s) = \alpha \times DI(i,j) - \beta \times DS(i,j,n) + \alpha \times DI(j+1,n)$$
$$= \alpha \times DI(i,j) - \beta \times DS(i,j,n) + D(j+1,n,1),$$
$$B(i,j,s) = \text{undefined}.$$

For a general s, where $2<s\leqslant K$, we have the following recursive formula if $n-j+1 \geqslant s$.

$$D(i,j,s) = \alpha \times DI(i,j) + \underset{j+1\leqslant k\leqslant n-s+2}{\text{minimum}}\{-\beta \times DS(i,j,k) + D(j+1,k,s-1)\},$$
$$B(i,j,s) = \text{a value of } k \text{ that gave the minimum in computing } D(i,j,s)$$

In the above equation, j is the first partition point, which is specified beforehand, and k is the second partition point, which needs to be selected. Splitting with these two points will partition the data into three parts: the first segment is from tuple i to tuple j, the second segment is from tuple $j+1$ to tuple k, and the remaining $s-2$ segments afterwards. For the first segment, there will be an intra-segment distance of $DI(i,j)$. Between the first and the second segments, there will be an inter-segment distance of $DS(i,j,k)$. Finally, for the data from the second segment until the last segment, the definition should ensure that its distance will be optimal. So, we use a cost term of $D(j+1,k,s-1)$. Since we are not sure which value of k gives the optimal partitioning point, we find it by trying all possible values in the range from $j+1$ to $n-s+2$. If there is a tie, we choose the smallest such k. Finally, the optimal distance can be determined as $\underset{2\leqslant j\leqslant n-K+1}{\text{minimum}}\{D(1,j,K)\}$.

Table 7
$D(i,j,s)$ matrix for the data in Table 1

$s=1$	1	2	3	4	5	6	7	8	9	10
1										
2										
3										
4										11
5										10
6										10
7										10
8										4
9										3
10										0

$s=2$	1	2	3	4	5	6	7	8	9	10
1										
2										
3			−33	−23	12	12	−18	−18	−33	
4				−38	12	12	−18	−18	−33	
5					−30	−25	−45	−15	−30	
6						−30	−50	−15	−30	
7							−60	−15	−30	
8								−17	−22	
9									−24	
10										

$s=3$	1	2	3	4	5	6	7	8	9	10
1										
2		−113	−68	−71	−75	−93	−50	−57		
3			−77	−98	−72	−90	−55	−57		
4				−113	−72	−90	−55	−57		
5					−70	−79	−82	−54		
6						−76	−87	−54		
7							−97	−54		
8								−40		
9										
10										

$s=4$	1	2	3	4	5	6	7	8	9	10
1	−137	−151	−143	−113	−112	−130	−73			
2		−166	−143	−113	−112	−130	−73			
3			−137	−140	−109	−127	−78			
4				−155	−109	−127	−78			
5					−103	−116	−105			
6						−113	−110			
7							−120			
8										
9										
10										

Based on the above equations, we give the following algorithm to perform the required computations.

Procedure: MGK(DI, DS, α, β, K) // Mining Generalized Knowledge

Begin
  $s \leftarrow 1$ //case 1: $s=1$
  $j \leftarrow n$
  For $i \leftarrow 1+K-s$ to $n-s+1$ do
   $D(i,j,s) \leftarrow \alpha \times DI(i,j)$
  End
  $s \leftarrow 2$ //case 2: $s=2$
  $k \leftarrow n$
  For $i \leftarrow 1+K-s$ to $n-s+1$ do
   For $j \leftarrow i$ to $n-s+1$ do
    $D(i,j,s)=\alpha\times DI(i,j)-\beta\times DS(i,j,k)+D(j+1,k,s-1)$
   End
  End
  For $s \leftarrow 3$ to K do //case 3: $2<s\leqslant K$
   For $i \leftarrow 1+K-s$ to $n-s+1$ do
    For $j \leftarrow i$ to $n-s+1$ do
     $D(i,j,s) = \alpha \times DI(i,j) + \underset{j+1\leqslant k\leqslant n-s+2}{\text{minimum}} \{-\beta \times DS(i,j,k) + D(j+1,k,s-1)\}$
     $B(i,j,s)$=a value of k that gave the minimum in computing $D(i,j,s)$
    End
   End
  End
End;

Example 4. Let us set $\alpha=1$, $\beta=4$ and $K=4$, and consider the data of Table 1 again. By applying the above algorithm, we can find the optimal distance matrix $D(i,j,s)$, as shown in Table 7. Meanwhile, the partition position matrix $B(i,j,s)$ is also obtained, as shown in Table 8.

Table 8
$B(i,j,s)$ matrix for the data in Table 1

$s=3$	1	2	3	4	5	6	7	8	9	10
1										
2		3	4	7	7	7	8	9		
3			9	7	7	7	8	9		
4				7	7	7	8	9		
5					9	7	8	9		
6						7	8	9		
7							8	9		
8								9		
9										
10										

$s=4$	1	2	3	4	5	6	7	8	9	10
1	2	4	4	6	7	7	8			
2		4	4	6	7	7	8			
3			4	6	7	7	8			
4				6	7	7	8			
5					7	7	8			
6						7	8			
7							8			
8										
9										
10										

3.6. The fifth phase

Finally, the last phase computes a set of partition points that achieves the optimum distance, denoted as $D(1,j^*,K)$, and outputs the generalized knowledge. We go repetitively backward from $B(1,j^*,K)$ and derive the generalized tuples at the same time. Let $GenSO(i,j)$ be the operation that outputs the generalized tuple summarized from tuple i to tuple j along with the count. The algorithm for outputting the knowledge is given below.

Procedure: CGK (D,B) // Create Generalized Knowledge

```
Begin
  //case 1: find the first and second partition points and the generalized tuples
  GenSO(1,j*)
  j ← B(1, j*, K)
  i ← j* + 1
  GenSO(i,j)
  //case 2: find the other partition points and the generalized tuples
  s ← K − 1
  While s > 2 Do
   k ← B(i, j, s)
   GenSO(j + 1,k)
   i ← j + 1
   j ← k
   s ← s − 1
  End
  GenSO(j + 1,n)
End;
```

Example 5. In the above example, the optimal distance is achieved when $D(1,2,4)=-151$, where we have $B(1,2,4)=4$. From this we know that the first segment is from tuple 1 to tuple 2 and the second segment from tuple 3 to tuple 4. After that, we set $i=3$, $j=4$ and $s=3$, and executing the while loop finds the result $B(3,4,3)=7$, which indicates that the third segment is from tuple 5 to tuple 7. Next, we exit the while loop because of $s=2$. Finally, the last step finds the fourth segment as from tuple 8 to tuple 10.

3.7. The time complexity

Let n denote the number of tuples in the input data, m the number of attributes, K the number of partitioned segments and h the maximum height of any concept tree. We assume that $m<n$ and $h<n$. Then the following are the time complexity of each phase and the total time complexity.

1. The complexity of phase 1, i.e., computing $F(i,r)$ for $1 \leqslant i \leqslant n-1$ and $1 \leqslant r \leqslant m$, can be computed as follows:

   $$(n-1) \times m \times h \in \theta(n \times m \times h).$$

2. The complexity of phase 2, i.e., computing $E(i,j,r)$ for $1 \leqslant i \leqslant j \leqslant n$ and $1 \leqslant r \leqslant m$, can be computed as follows:

$$\sum_{r=1}^{m}\sum_{i=1}^{n-1}(n-1-i-1+1) = m \times \left(\frac{n^2-3n}{2}+1\right) \in \theta(n^2 \times m).$$

3. The complexity of phase 3, i.e., computing $DI(i,j)$ for $1 \leqslant i < j \leqslant n$, can be computed as follows:

$$\sum_{i=1}^{n}\sum_{j=i}^{n} m \in \theta(n^2 \times m).$$

4. The complexity of the first job in phase 4, i.e., computing $DS(i,j,k)$ for $1 \leqslant i \leqslant j < k \leqslant n$, can be computed as follows:

$$\sum_{i=1}^{n-1}\sum_{j=i}^{n-1}\sum_{k=j+1}^{n} 1 \in \theta(n^3).$$

5. The complexity of the second job in phase 4, i.e., computing $D(i,j,s)$ for $1 \leqslant i \leqslant j \leqslant n$ and $1 \leqslant s \leqslant K$, can be computed as follows:

$$\sum_{s=1}^{K}\sum_{i=1}^{n}\sum_{j=i}^{n} n \in \theta(n^3 \times K).$$

6. The complexity of phase 5, i.e., determining how to partition the data, is $\theta(K)$ since we go backward from $B(1,j^*,K)$ in K steps to find all partition points. After finding all partition points, the time to produce the K generalized tuples can be done in time $\theta(n \times m \times h)$.
7. Summing all the above time complexities together, we have the final time complexity as $\theta(n^3 \times K)$.

The final time complexity $\theta(n^3 \times K)$ is much better than the naïve time complexity $\theta(C_K^n \times n \times m \times h) = \theta(n^{K+1} \times m \times h)$, which is achieved by trying all possible partitions and doing generalizations for each partition. However, it is significantly worse than AOI's $\theta(n \times m \times h)$ complexity.

3.8. The space complexity

In this section, we will analyze how much space is required to store all the data structures used in the algorithm. In the following, we list the space requirements for each phase.

- Phase 1 uses table $F(i,r)$, where i is from 1 to $n-1$ and r is from 1 to m. Thus, the space required is $\theta(n \times m)$.
- Phase 2 uses table $E(i,j,r)$, where $1 \leqslant i \leqslant j \leqslant n$ and $1 \leqslant r \leqslant m$. Thus, the required space is $\theta(n^2 \times m)$.
- Phase 3 uses table $DI(i,j)$, where $1 \leqslant i < j \leqslant n$. Thus, the space required is $\theta(n^2)$.
- Phase 4 uses table $DS(i,j,k)$, where $1 \leqslant i \leqslant j < k \leqslant n$. Thus, the space required is $\theta(n^3)$.
- Phase 4 uses two other tables D (i,j,s) and $B(i,j,s)$, where $1 \leqslant i \leqslant j \leqslant n$ and $1 \leqslant s \leqslant K$. Thus, the space required is $\theta(n^2 \times K)$.
- The total space requirements are $\theta(n^3)$.

We can reduce the above space complexity to $\theta(n^2 \times \max\{K,m\})$. The table DS occupies the most space, but each $DS(i,j,k)$ value equals to $DI(i,k) - DI(i,j) + DI(i,k) - DI(j+1,k)$. Instead of actually creating this table, when we need the value of $DS(i,j,k)$, we can directly compute this value based on this formula. This approach reduces the space required for storing the table to $\theta(n^2 \times \max\{K,m\})$.

4. Extensions to the Ordered AOI Algorithm

In this section, we will discuss several difficulties that we may encounter when using the Ordered AOI algorithm to mine real data. For each of these difficulties, a solution is proposed as well. After including these extensions, the resulting algorithm is called the Extended Ordered AOI algorithm (the EOAOI algorithm for short). The following are the difficulties that we may encounter.

1. Our method uses a small number of generalized tuples to represent a large number of ordered data. Therefore, if a generalized tuple is formed by grouping only a few tuples, we may feel that the basis of this generalized tuple is inadequate because it only represents a small group of data; this results in a generalized tuple of weak representation. In view of this, we add another constraint that the number of tuples summarized into a generalized tuple must be no less than *lb* and no greater than *ub*.
2. A key element in the AOI method is concept trees. However, when we try to apply our algorithm to real data, especially numerical data, we found that the concept tree cannot adequately deal with numerical data. For example, let us build a concept tree for the score ranging from 0 to 100. Suppose that the root $[0, 100]$ has two children at the first level, where the left son is $[0, 50)$ and the right son is $[50, 100]$. Then, if we have two numerical data 49 and 51 in the same group, we must generalize them into the most top level $[0, 100]$ of the tree, though their difference is only 2. To remedy this problem, we propose using a directed acyclic graph structure to solve this problem.
3. In practice, noise and outliers commonly occur in datasets. Without dealing with these dirty data, the generalization result may be distorted because of these extreme values. Therefore, we propose a method to preprocess the data so that the noise can be removed.
4. The Ordered AOI algorithm has a high time complexity of $\theta(n^3)$, which makes it impractical for large data sets. We propose a preprocessing method that can reduce the data size and thus improve the performance.

In the following three subsections, we will describe the proposed extensions.

4.1. Size constraint

We extend the ordered AOI algorithm by adding the constraint that the number of tuples summarized into a generalized tuple must be no less than *lb* and no greater than *ub*. Basically, we employ a similar procedure like the original algorithm to solve the new problem. The main differences lie in how we define the table D and execute phase 4.

Let $D(i,j,s)$ denote the minimum total distance of the data from tuple i to the last tuple if we partition it into s segments and the first segment is from tuple i to tuple j. Because of the newly added constraint, we require $lb \leqslant j-i+1 \leqslant ub$. Complementarily, $B(i, j, s)$ denotes the first partition point after tuple j that achieves the optimum distance of $D(i,j,s)$, where $lb \leqslant j-i+1 \leqslant ub$. If $s=1$, we have the following base relation:

$$D(i,j,s) = \alpha \times DI(i,n) \text{ if } j = n \text{ and } lb \leqslant n-i+1 \leqslant ub,$$

$$D(i,j,s) = \text{undefined if } j \neq n,$$

$$B(i,j,s) = \text{undefined}.$$

In the case that $s=2$, $lb \leqslant j-i+1 \leqslant ub$, $lb \leqslant n-j \leqslant ub$ and $n>j$, the recursive relation is as follows:

$$\begin{aligned} D(i,j,s) &= \alpha \times DI(i,j) - \beta \times DS(i,j,n) + \alpha \times DI(j+1,n) \\ &= \alpha \times DI(i,j) - \beta \times DS(i,j,n) + D(j+1,n,1), \\ B(i,j,s) &= \text{undefined}. \end{aligned}$$

For a general s, where $2<s \leqslant K$, we have the following recursive formula if $n-j+1 \geqslant s$, $lb \leqslant j-i+1 \leqslant ub$, $lb\times(s-1) \leqslant n-j \leqslant ub\times(s-1)$:

$$D(i,j,s) = \alpha \times DI(i,j) + \underset{j+lb \leqslant k \leqslant \min\{n-(s+2)\times lb,\, j+ub\}}{\text{minimum}} \{-\beta \times DS(i,j,k) + D(j+1,k,s-1)\},$$

$$B(i,j,s) = \text{a value of } k \text{ that gave the minimum in computing } D(i,j,s).$$

Based on the above formulas, the MGK algorithm in Section 3.5 can be modified to give the extended MGK algorithm. To save space, we omit the details.

Sometimes we may only have to set the lower bound lb but leave the upper bound ub unlimited. Our method can easily handle this special case just by setting $ub=n-(K-1)\times lb$.

4.2. Common-child tree

As mentioned, using the concept tree to represent numerical data may face a serious situation when the data in the same group fall into the boundary of two adjacent regions. To remedy, we slightly modify the original concept tree as a new structure, called the common-child tree. The common-child tree is similar to the concept tree. It still has multiple levels of nodes, and each node belongs to a certain level. The difference lies in that in a common-child tree, if node A is on the left adjacent to node B, then (1) all the child nodes of node A must be on the left of those of node B, and (2) the rightmost child of node A and the leftmost child of node B can be the same node. But in a concept tree two nodes at level $h-1$ are not allowed to have any common child at level h. Fig. 1 shows an example of a common-child tree for the score ranging from 0 to 100.

When we use the common-child tree rather than the concept tree, the original algorithm should be modified accordingly. In this new structure, the most important difference lies in that there are multiple paths that a leaf node can go to the root node, while in the original concept tree there is only one path that we can go from a leaf node to the root node. Because of this difference, phase 1 of the original algorithm is no longer appropriate, because Lemma 1 no longer holds and this means we cannot compute the $E(i,j,r)$ table from the $F(i,r)$ table. So, our problem becomes finding how we can compute the $E(i,j,r)$ table.

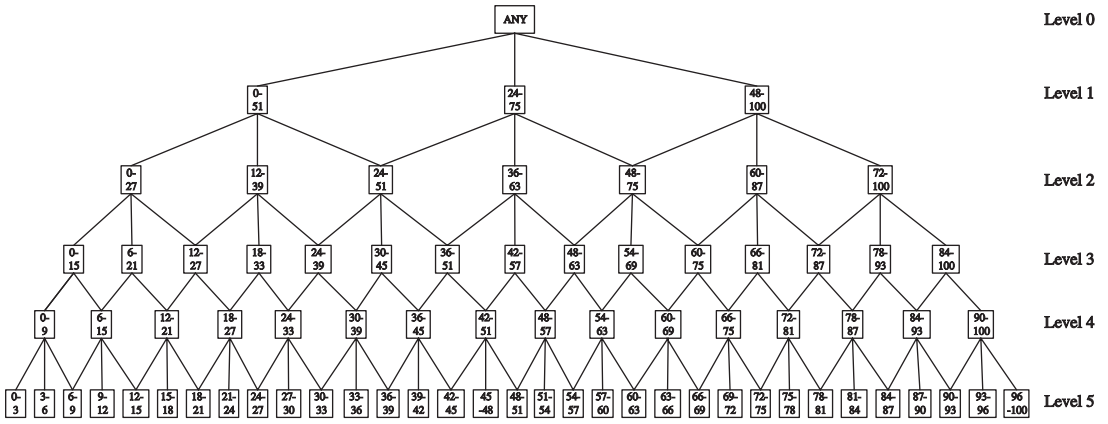

Fig. 1. The common-child tree for the score ranging from 0 to 100.

Let us consider the common-child tree of attribute r. First, we traverse all the paths from the root node to every leaf node in the tree. Each time we reach a leaf node, we store the information including the nodes along the path as well as their levels into the node. After the traversal is over and duplicate information is removed, each leaf node will record all its ancestor nodes along with their levels. Let node $l(w)$ denote the leaf node to which the value of attribute r of tuple w belongs. Let $A(l(w))$ denote the set of ancestor nodes of node $l(w)$, and let $level(u)$ denotes the level of node u for each node u in $A(l(w))$. Then the nearest common ancestor z of nodes $l(w)$ for $i \leqslant w \leqslant j$ can be determined as follows:

$$U_{i,j} = \{u \text{ exist in all } A(l(w)) \text{ for } i \leqslant w \leqslant j\},$$
$$level(z) = \min_{u \in U_{i,j}} \{level(u)\}.$$

A quicker way to determine the common ancestor is to sort all elements in $A(l(w))$ into a non-ascending sequence first by their levels and then by their names. Then we can find the common ancestor z by sequentially scanning and comparing all these sequences until the first common element appears. Having found z, the value of $E(i,j,r)$ is determined as $L_r - level(z)$.

In fact, $E(i,j,r)$ can be computed from $E(i,j-1,r)$. The relation $U_{i,j} = U_{i,j-1} \cap A(l(j))$ indicates that if we already know $U_{i,j-1}$, then $U_{i,j}$ can be determined by merging set $U_{i,j-1}$ with $A(l(j))$. After the merge, the first one appearing in $U_{i,j}$ is the nearest common ancestor with the minimum level. Therefore, we use the following procedure to compute the $E(i,j,r)$ table.

```
For i ← 1 to n − 1 Do
    U_{i,i} = A(l(i))
    For j ← i + 1 to n Do
        U_{i,j} = U_{i,j−1} ∩ A(l(j))
        z = the first element in U_{i,j}
        E(i,j,r) = L_r − level(z)
    End
End
```

After determining the $E(i,j,r)$ table, the remaining parts of the algorithm are not influenced by the common-child tree, except for the operation $GenSO(i,j)$ used in the CGK algorithm in Section 3.6. This operation finds the generalized tuple for the data from tuple i to tuple j. This can be done by finding their nearest common ancestors for all attributes. We apply the above method to do so.

4.3. Data preprocessing

In this section, we will discuss how to preprocess the data so that two aims can be achieved: (1) noise can be removed from the data, and (2) the data size can be reduced and the performance can be improved. If no preprocessing is used, we may face two difficulties.

- If there exist some tuples in a group which do not comply with the general behavior of the data in that group, the generalization will yield uninteresting results, because the tuple summarized from that group will be too general to be meaningful. For example, if we are told that the generalized tuple has the value "ANY" in every column, this information seems no value for us.
- The analysis in Section 3.7 shows a time complexity of $\theta(n^3)$ for our algorithm but the traditional AOI algorithms only requires time $\theta(n \times m \times h)$. This suggests that our algorithm is not suitable for dealing with large databases.

Since the aim of our method is to find how the characteristics change along the dimension according to which we sort the data, what is important is the trend along this dimension rather than every single detail. On the other hand, when the data have noise, the generalization process may collapse if we really consider every single detail. In view of these two concerns, we use the following data reduction method to preprocess the data, in hope that they can improve the efficiency without losing the important trend of the data.

We reduce the number of tuples to $1/R$ of the original number, and use representatives to represent the original data. To this end, we first partition the data list into $\lceil n\ R \rceil$ consecutive segments, each of which has R tuples, except for the last one. Then, we use a representative tuple to represent each segment of data. We choose the representative value for different types of attributes as follows.

- If the attribute is numerical, we use the mean value of these R tuples as the representative value. If the mean is not itself a legal value, then the median would need to be used instead.
- If the attribute is not numerical but it is ordered, we use the median of these R tuples as the representative value.
- If the attribute is unordered, we use the mode of these R tuples as the representative value.

5. Performance evaluations

In this section, we perform a simulation study to empirically evaluate the performance of the proposed method. The EOAOI algorithm is implemented in the Visual Basic 6.0 language and tested on a PC with a P4 2.4G processor and 1024MB main memory under the Windows 2000 operating system. Note that the algorithm implemented includes all the extensions mentioned in Section 4.

Since the AOI method has a much better time complexity than the EOAOI algorithm, the objective of this simulation study is not to compare the run times of the two algorithms, but it is instead to identify the maximum size of data set that the EOAOI algorithm can process in a reasonable time. Three factors are considered in the study: (1) the number of data records, (2) the size of R, and (3) the relative ratio between *ub* and *lb*. For each factor, we study how the run time varies as we change the value of the factor. After this study, the next experiment is to study whether or not the preprocessing step is worth performing. This evaluation is performed from the perspective of the output quality. What we are interested is if this preprocessing can help us to remove the noise data and hence improve the quality of the generalization. We compare the output of our algorithm for the following cases: (1) $R=1$, (2) $R=2$, (3) $R=10$, and (4) $R=50$.

In the following, we first explain how the test data are generated and how the parameters are set. Then we discuss the two experiments mentioned above, where the first is concerned with run time and the second is concerned with the quality of the output of the algorithm.

5.1. Data generation

To study the performance of the proposed algorithm, we randomly generate the synthetic data sets for the student streaming exams. We assume that there are six subjects that the students need to be tested, and that the sum of the scores in these six subjects is used to rank the students, where the score s_i of subject i is ranged from 0 to 100, where $1 \leqslant i \leqslant 6$. We assume that the score of each subject complies with a normal distribution, and Table 9 shows the mean and the standard deviation of each subject. The six pairs of values shown in Table 9 are borrowed from the public data reported by The Ministry of Education (MOE), Taiwan in year 2003, which holds nationwide streaming exams every year.

Table 9
The mean and the standard deviation of each subject

Subjects	$s1$	$s2$	$s3$	$s4$	$s5$	$s6$
Mean	50	43	32	46	36	57
Standard deviation	16	23	21	24	20	22

We use the following steps to generate a synthetic data tuple for each simulated student. First, we randomly draw a real number x in $[0,1]$ to denote the overall performance of this student. Then for each subject, say subject i, we randomly draw a real number y_i in $[0,1]$ to denote how this student performs on subject i. Let a_i denote the accumulated percentage of scores of subject i that are no better than that of this student. Then a_i is determined as $a_i = w \times x + (1-w) \times y_i$, where w is a given constant in $[0,1]$. Throughout the experiments, we assume that $w = 0.6$. Finally, based on a_i and the mean and the standard deviation of subject i we can determine the score s_i of this student.

We use the tree in Fig. 1 as the common-child tree for each attribute. We assume that all attributes are of equal importance, and we set $\alpha = 1$ and $\beta = 100$. Let N denote the number of tuples and let r denote the constraint on the ratio of ub/lb. Then, we define ub as $(N/K) \times \sqrt{r}$ and lb as $(N/K)/\sqrt{r}$.

5.2. Run time

This section compares the run time for two algorithms, i.e., the AOI algorithm and the EOAOI algorithm. First, we evaluate the size of data set that our algorithm can process in a reasonable time. For the EOAOI algorithm, we set $R = 1$ and fix the ratio of ub/lb as 4, where $R = 1$ means that no preprocessing is done. Besides, our program is run by setting three different values of K, i.e., $K = 10$, $K = 15$ and $K = 20$. As for the AOI algorithm, we build a concept tree for the score by repeatedly partitioning the intervals into two equal-width subintervals until the depth of the tree is five. When running the AOI, we assume that all data tuples will be generalized to the top level. The simulation is carried out by varying the number of tuples, denoted as N, from 500 to 2500. The results are shown in Fig. 2. Obviously, the AOI method is faster than the EOAOI algorithm. However, we find that the EOAOI algorithm can process several thou-

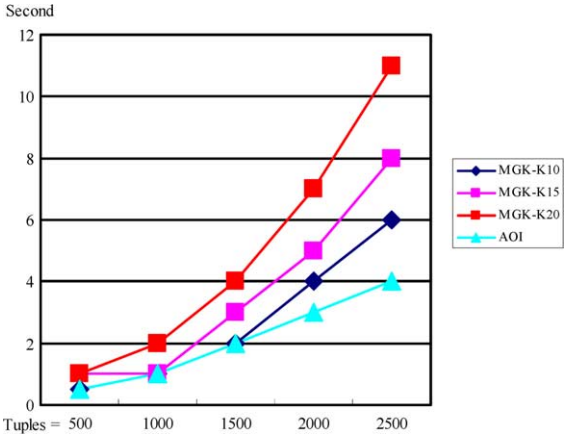

Fig. 2. Run time vs. data size.

sands data tuples in a few seconds. This result indicates that when the sizes of data sets are not too large, the EOAOI algorithm would be capable of solving them.

Next, we will study how the number of generalized tuples influences the run time of the EOAOI algorithm. To this end, we set $R=1$, $N=2500$ and fix the ratio of ub/lb as 4. Then K was varied from 5 to 25. Since the AOI method does not use this parameter, we do not include the AOI method in this test. The results are shown in Fig. 3. From this figure, we find that the run time increases as the size of K gets larger. But a problem arising immediately in the figure is why the run time increases sharply when the value of K is larger than 25. After a careful examination, we found that it is not resulted from the algorithm itself. But it is because, when K is large, the memory space is not enough to keep the stored tables. Thus, the operating system has to swap the tables between the main memory and the disk to keep the program executable in a limited space.

Finally, we will study how the ratio between ub and lb influences the run time of the algorithm. To do so, we set $R=1$, $N=2500$ and $K=10$. The value of ub/lb was varied from 1.5 to 6. The results are shown in Fig. 4. From this figure, we find that the run time increases as the ratio gets larger. This result indicates that adding the constraint on the ratio of ub/lb brings about two advantages. First, it can improve the quality of the generalization, because it ensures that every generalized tuple is derived from enough data. Second, it reduces the run time.

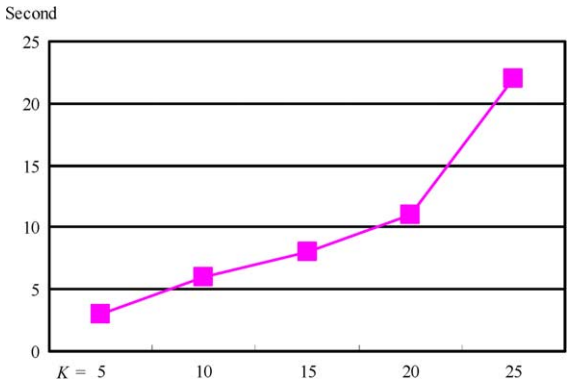

Fig. 3. Run time vs. the number of generalized tuples.

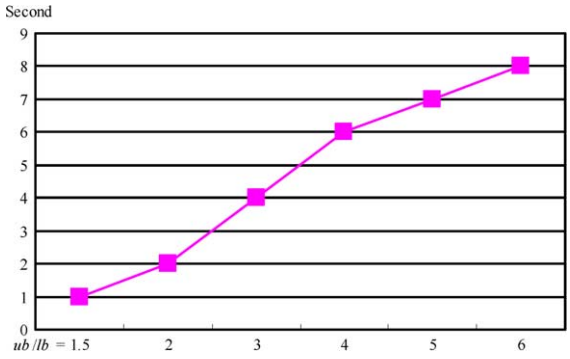

Fig. 4. Run time vs. the ratio of ub and lb.

5.3. The size of the smoothing window

The above section shows how the run time changes as we change the values of parameters. In this section, we show what generalized results will be produced when we varied the size R of the smoothing win-

Table 10
The output for: (a) $R=1$; (b) $R=2$; (c) $R=10$ and (d) $R=50$

$s1$	$s2$	$s3$	$s4$	$s5$	$s6$	count
Panel (a)						
48–100	ANY	ANY	48–100	ANY	ANY	158
48–100	ANY	ANY	ANY	24–75	ANY	100
ANY	ANY	24–75	ANY	24–75	ANY	399
24–75	24–75	24–75	24–75	24–75	48–100	100
24–75	24–75	ANY	24–75	ANY	ANY	399
36–63	24–75	0–51	24–75	24–51	24–75	100
24–75	ANY	0–51	ANY	0–51	24–75	399
24–75	0–51	0–51	ANY	0–51	36–63	100
ANY	0–51	0–51	ANY	0–51	ANY	399
0–51	0–51	0–51	0–51	0–51	ANY	346
Panel (b)						
48–100	48–100	24–75	48–100	ANY	48–100	185
48–75	24–75	24–75	48–75	36–63	48–100	50
24–75	24–75	24–51	24–75	24–75	48–100	199
36–63	36–63	24–51	36–63	24–51	48–75	50
36–63	24–75	0–51	24–75	24–51	48–75	199
36–63	24–51	12–39	24–75	24–51	24–75	50
24–75	0–51	0–51	ANY	0–51	24–75	199
36–51	0–51	0–51	0–51	12–39	36–63	50
24–51	0–51	0–51	0–51	0–51	24–75	199
0–51	0–51	0–27	0–51	0–51	0–51	69
Panel (c)						
60–75	48–100	48–63	48–100	48–75	72–87	10
48–75	48–75	36–63	48–75	36–63	48–100	40
48–63	48–57	36–45	54–63	42–51	60–75	10
48–63	42–57	30–45	36–63	36–51	48–75	39
48–57	42–51	30–39	42–57	36–45	54–63	10
42–57	36–51	24–39	36–51	30–45	48–63	39
42–51	30–45	24–33	36–51	24–39	48–57	10
36–51	24–51	18–33	30–45	24–39	42–57	39
36–45	24–39	18–27	30–39	18–33	36–51	13
24–51	0–51	0–27	0–51	0–51	24–51	40
Panel (d)						
60–69	48–75	42–57	60–75	48–63	60–87	6
54–63	57–60	42–51	54–63	45–48	66–75	2
54–63	48–57	36–45	48–63	42–51	60–69	7
54–57	48–51	36–39	51–54	36–45	63–66	2
51–54	42–51	30–39	42–57	36–45	54–63	7
48–51	39–42	30–33	45–48	30–39	57–60	2
42–51	36–45	24–33	42–45	24–39	48–57	7
45–48	36–39	24–33	36–45	27–30	48–57	2
36–51	24–39	18–27	30–39	24–33	42–51	7
30–45	12–39	6–21	12–39	12–27	30–45	8

dow. In this test, we output the generalized results for $R = 1$, 2, 10, and 50. The results are shown in Table 10(a)–(d), respectively.

From these four output results, we find that the preprocessing step is critical for the success of the proposed algorithm. Without preprocessing (Table 10(a)), noise in the data may severely bias the generalized results so that many ANY values appear in the output. This makes the output uninteresting for the users. If the preprocessing step is performed, the output shows the trend in the data. However, we should not use a very large value for R when preprocessing the data. For example, let us observe the first tuple and the last tuple in Table 10(d), where $R = 50$. We find that those scores which are too high are excluded from the score range in the first tuple. Similarly, those scores which are too low are also excluded from the score range in the last tuple.

6. Conclusion

Since the current AOI-based methods can only find generalized knowledge from relational data and since ordered data occur frequently in practice, we considered the possibility of finding generalized knowledge from an ordered list of data by modifying the AOI method. In the paper, we proposed a dynamic-programming based algorithm, with which we can discover a sequence of K generalized tuples describing the characteristics of different segments of data along the list. The time complexity of the algorithm is $\theta(n^3 \times K)$.

In addition, we discussed how to deal with four potential problems that may be encountered when applying the proposed algorithm to real data sets. To remedy these four problems, three extensions of the algorithm were proposed, including (1) adding a constraint on the ratio between *ub* and *lb*, (2) constructing common-child trees for the attributes other than concept trees, and (3) preprocessing the data.

In the simulation, we applied the extended algorithm to the synthetic data sets. From the experimental results, we found that the extended algorithm running on a typical PC can process thousands of tuples in few seconds. Thus, our algorithm can be applied in many real data sets which are not too large. Furthermore, our simulation also shows that the preprocessing step not only improves the output quality but also reduces the data size. This finding indicates that our algorithm can deal with a large data set if the data set can be preprocessed properly.

Some possible future research directions for this research are as follows.

1. As given, the proposed algorithm is not suitable for very large datasets, such as commonly occur in data mining. A significant limitation of our approach is that the time complexity is $\theta(n^3 \times K)$ while standard AOI is $\theta(n)$. Thus, future research could attempt to propose improved algorithms with sufficient efficiency to discover generalized knowledge in very large ordered data sets.
2. Several extensions of AOI have been proposed since its introduction in 1990. For ordered data, we could consider variations such as associating multiple concept trees with a single attribute, using domain generalization graphs rather than concept trees (Hamilton et al., 1996), using fuzzy concept trees rather than crisp concept trees and using rough set theory to further remove redundant attributes (Hu and Cercone, 1996).
3. In addition, we may consider the extension of how we can apply the AOI technique to induce knowledge from types of data other than sets or lists, say from graph data, acyclic graph data or tree data. Additionally, we may apply the AOI technique to data of mixed types or a universal type.
4. Instead of generalizing the data only based on a single ordering, another possible extension is to do the generalization based on multiple orderings, because multi-attribute data can support the definition of different orderings for different attributes.

5. An important contribution of this paper is that we define the distance between generalized tuples, based on which optimization algorithms can be developed. This new way of applying AOI extends the applicability of the AOI techniques because AOI can be used to search for optimum solutions to optimization problems. In the future, researchers could attempt to define other distance measures and apply AOI to solve other optimization problems.
6. To support decision making a decision maker must gather information by repeatedly analyzing the data, each time from a different perspective. During this iterative process, the environment may be constantly changing. Therefore, running our algorithm from the scratch every time the environment changes would be time consuming. In view of this weakness, future research could attempt to design incremental algorithms that produce generalized knowledge by starting from the previous result.

Acknowledgement

The research was supported in part by the MOE Program for Promoting Academic Excellence of Universities under the Grant Number 91-H-FA07-1-4. We would like to thank anonymous referees for their helpful comments and suggestions.

Appendix A

See Figs. A.1–A.4.

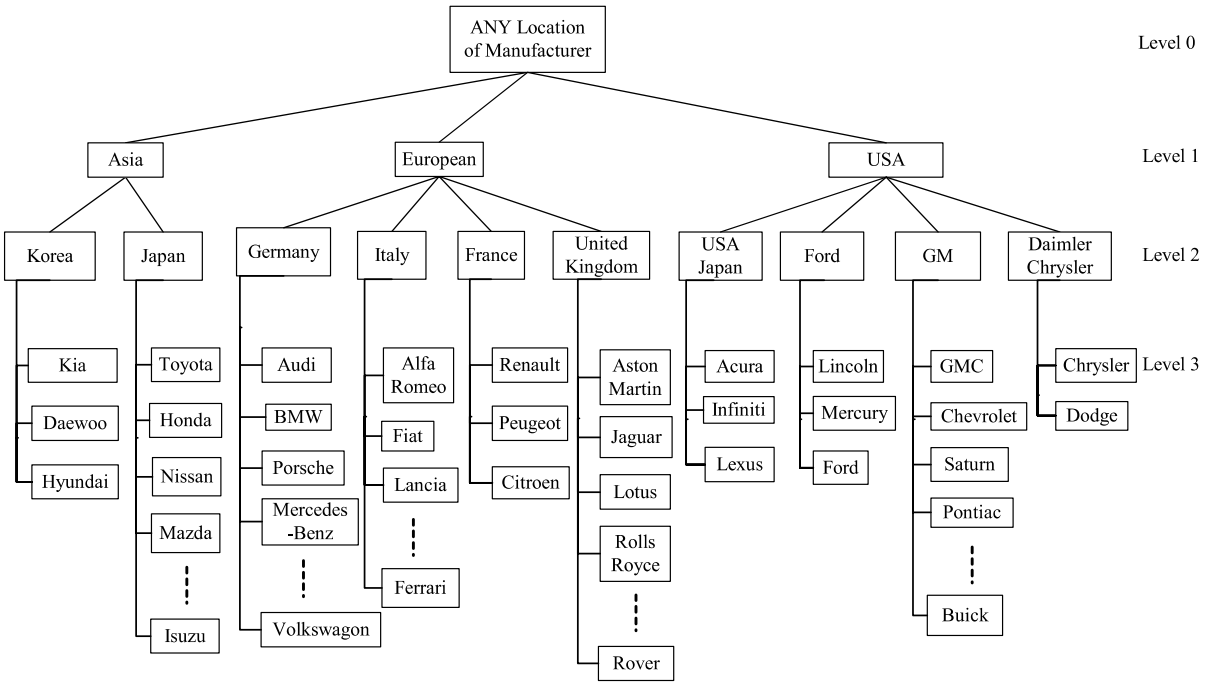

Fig. A.1. The concept tree for attribute "Location of Manufacturer".

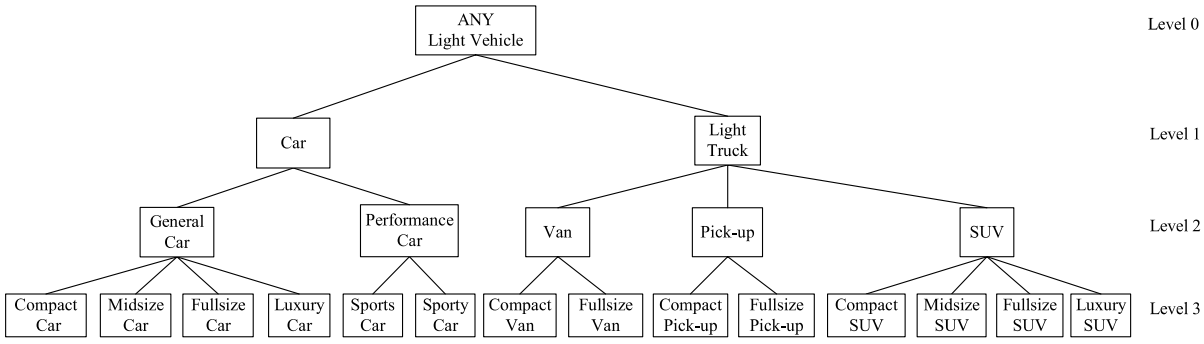

Fig. A.2. The concept tree for attribute "Light Vehicle Model".

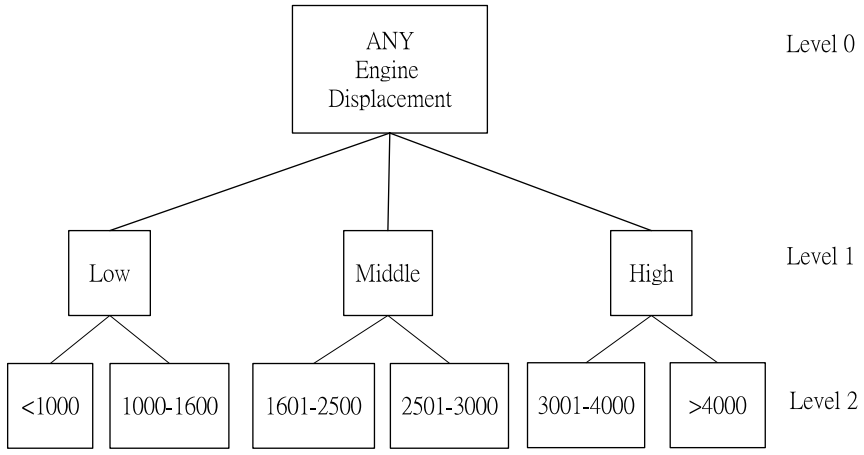

Fig. A.3. The concept tree for attribute "Engine Displacement".

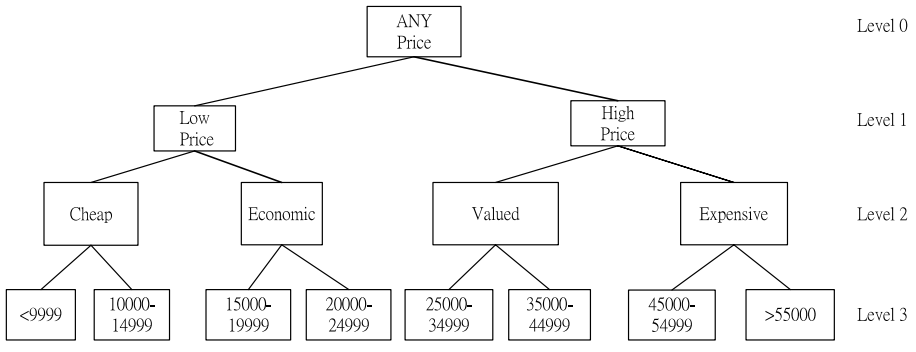

Fig. A.4. The concept tree for attribute "Price".

References

Cai, Y., Cercone, N., Han, J., 1990. An attribute-oriented approach for learning classification rules from relational databases. In: Proceedings of Sixth International Conference on Data Engineering, pp. 281–288.

Carter, C.L., Hamilton, H.J., 1995. Performance evaluation of attribute-oriented algorithms for knowledge discovery from databases. In: Proceedings of Seventh International Conference on Tools with Artificial Intelligence, pp. 486–489.

Carter, C.L., Hamilton, H.J., 1998. Efficient attribute-oriented generalization for knowledge discovery from large databases. IEEE Transactions on Knowledge and Data Engineering 10 (2), 193–208.

Chen, M.S., Han, J., Yu, P.S., 1996. Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering 8 (6), 866–883.

Cheung, D.W., Hwang, H.Y., Fu, A.W., Han, J., 2000. Efficient rule-based attribute-oriented induction for data mining. Journal of Intelligent Information Systems 15 (2), 175–200.

Hamilton, H.J., Hilderman, R.J., Cercone, N., 1996. Attribute-oriented induction using domain generalization graphs. In: Proceedings of Eighth IEEE International Conference on Tools with Artificial Intelligence, pp. 246–252.

Han, J., Kamber, M., 2001. Data Mining: Concepts and Techniques. Academic Press, New York.

Han, J., Cai, Y., Cercone, N., 1992. Knowledge discovery in databases: An attribute-oriented approach. In: Proceedings of International Conference on Very Large Data Bases (VLDB-92), pp. 547–559.

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

Han, J., Nishio, S., Kawano, H., Wang, W., 1998. Generalization-based data mining in object-oriented databases using an object-cube model. Data and Knowledge Engineering 25, 55–97.

Hu, X., Cercone, N., 1996. Mining knowledge rules from databases: A rough set approach. In: Proceedings of the Twelfth International Conference on Data Engineering, pp. 96–105.

Lu, W., Han, J., Ooi, B.C., 1993. Discovery of general knowledge in large spatial databases. In: Proceedings of 1993 Far East Workshop on Geographic Information Systems (FEGIS-93), pp. 275–289.

McClean, S., Scotney, B., Shapcott, M., 2000. Incorporating domain knowledge into attribute-oriented data mining. International Journal of Intelligent Systems 15 (6), 535–548.

Shan, N., Hamilton, H.J., Cercone, N., 1995. GRG: Knowledge discovery using information generalization, information reduction, and rule generation. In: Proceedings of the Seventh International Conference on the Tools with Artificial Intelligence, pp. 372–379.

Tsumoto, S., 2000. Knowledge discovery in clinical databases and evaluation of discovered knowledge in outpatient clinic. Information Sciences 124 (1), 125–137.