Published in Proc. of Int. Conf. on Extending Database Technology, Valencia, Spain, 1998.

Incremental Generalization for Mining in a Data Warehousing Environment

Martin Ester, Rüdiger Wittmann

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 Muenchen, Germany
{ester | wittman}@informatik.uni-muenchen.de
http://www.dbs.informatik.uni-muenchen.de

Abstract. On a data warehouse, either manual analyses supported by appropriate visualization tools or (semi-) automatic data mining may be performed, e.g. clustering, classification and summarization. Attribute-oriented generalization is a common method for the task of summarization. Typically, in a data warehouse update operations are collected and applied to the data warehouse periodically. Then, all derived information has to be updated as well. Due to the very large size of the base relations, it is highly desirable to perform these updates incrementally. In this paper, we present algorithms for incremental attribute-oriented generalization with the conflicting goals of good efficiency and minimal overly generalization. The algorithms for incremental insertions and deletions are based on the materialization of a relation at an intermediate generalization level, i.e. the anchor relation. Our experiments demonstrate that incremental generalization can be performed efficiently at a low degree of overly generalization. Furthermore, an optimal cardinality for the sets of updates can be determined experimentally yielding the best efficiency.

Keywords: Data Mining, Data Warehouses, Generalization, Database Updates.

1 Introduction

Many companies have recognized the strategic importance of the knowledge hidden in their large databases and have built data warehouses. A *data warehouse* contains information collected from multiple data sources, integrated into a common repository and extended by summary information (such as aggregate views) for the purpose of analysis [MQM 97]. Often, a data warehouse - used for online analytical processing - is implemented as a collection of materialized views, i.e. a (possibly extended) copy of the operational data bases - used for online transaction processing [Huy 97].

On a data warehouse, either manual analyses supported by appropriate visualization tools or (semi) automatic data mining may be performed. *Data mining* has been defined as the application of data analysis and discovery algorithms that - under acceptable computational efficiency limitations - produce a particular enumeration of patterns over the data [FSS 96]. Several data mining tasks have been identified [FSS 96], e.g. clustering ([EKX 95], [EKSX 96]), classification [HF 96] and summarization ([AS 94], [HCC 93]). Typical results of data mining are as follows:

- clusters of items which are typically bought together by some class of customers (clustering in a sales transactions data warehouse)
- rules discriminating disease A from disease B (classification in a medical data warehouse)
- description of the typical WWW access patterns (summarization in the data ware-

house of an internet provider).

In this paper, we assume the data warehouse to be implemented as a relational database which is a common approach.

Generalization is an important method for the task of summarization. A*ttribute-oriented generalization* [HCC 93] of a relation is the process of replacing the attribute values by a more general value, one attribute at a time, until the relation fulfills some termination condition, e.g. the number of its tuples becomes less than a threshold. The reduction of the number of tuples enables effective and efficient manual analyses as well as automatic data mining on the generalized relations [HF 96].

Typically, a data warehouse is not updated immediately when insertions and deletions on the operational databases occur. Update operations are collected and applied to the data warehouse periodically, e.g. each night [MQM 97]. Then, all patterns derived from the warehouse by some data mining algorithm have to be updated as well. This update has to be efficient enough to be finished before the operation of the warehouse starts again, e.g. in the next morning. Due to the very large size of the base relations, it is highly desirable to perform these updates incrementally [Huy 97], e.g. to consider only the old generalized relations and the tuples to be inserted or deleted, instead of applying the generalization algorithm to the (very large) updated base relation.

The problem of incrementally updating mined patterns on changes of the database has just started to receive some attention. [CHNW 96] describes an efficient method for incrementally modifying a set of association rules [AS 94] mined from a database. In [HCC 93], the incremental update of generalized relations is discussed but no algorithms are presented. The goal of this paper is to propose algorithms for incremental attribute-oriented generalization and to evaluate their performance both, in terms of efficiency and effectivity.

The rest of the paper is organized as follows. Section 2 introduces the concepts of attribute-oriented generalization and presents a new algorithm. In section 3, we discuss the requirements for incremental generalization. Section 4 presents incremental algorithms for insertion and deletion to a relation with derived generalizations. In section 5, we report the results of our performance evaluation. Section 6 summarizes the contributions of this paper and discusses several issues for further research.

2. Attribute-Oriented Generalization

A*ttribute-oriented generalization* [HCC 93] of a relation is the process of replacing the attribute values by a more general value, one attribute at a time, until the number of tuples of the relation becomes less than a specified threshold. The more general value is taken from a concept hierarchy which is typically available for most attributes in a data warehouse. In the following, we introduce some notions needed for a more formal specification.

Definition 1: card(R), d_i

Let R be a relation with attributes A_i, $1 \le i \le k$, *card(R)* be the number of tuples of R and d_i be the number of different values of A_i in R.

Definition 2: $[A_{i1}, , , A_{im}]$**,** $A_{ij}^{\,l}$

We denote the sequence of attributes selected for generalization (*attribute sequence* for short) by $[A_{i1}, , , A_{im}]$, A_{ij} ε $\{A_1, \ldots, A_k\}$ and *l consecutive selections* of attribute A_{ij} by $A_{ij}^{\,l}$.

Definition 3: concept hierarchy, ANY, height

The *concept hierarchy* of A_i, which is not mandatory, is a balanced tree and is denoted by C_i. Its nodes represent the possible values (concepts) of A_i and its edges represent the is-a (superterm) relation between the concepts of the connected nodes.
The root of each concept hierarchy contains the special concept *ANY* which is a superterm of all other values (concepts).
Let the *height* of the leaves of a concept hierarchy be 0. The *height* of an interior node of a concept hierarchy is the height of its direct children plus 1.

Definition 4: base relation, generalized relation, fully generalized relation

Let *R* be a relation. We call *R* a *base relation* if all attribute values are from leaves of their respective concept hierarchy, i.e. if no generalization has been performed.
R is called a *generalized relation* if it has been derived from some base relation by performing one or more generalizations. Each generalized relation has an additional attribute *support* counting the number of tuples of the base relation subsumed by the generalized tuple.
We call *R* a *fully generalized relation w.r.t. T*, denoted by R_T, if $card(R) \leq T$.

Definition 5: generalization level

Let *R* be a relation and A_i, $1 \leq i \leq k$, be one of its attributes.The *generalization level of* A_i, denoted as g_i, is defined as the height of the values of A_i in their respective concept hierarchy. The *generalization level of* *R*, denoted as G_R, is defined as:.

$$G_R = \sum_{i=1}^{k} g_i$$

The basic algorithm presented in [HCC 93] is presented in figure 1. This algorithm has some drawbacks. Typically, after the first phase *card(R)* is much larger than *T* and, in the worst case, even $card(R) = T^k$, i.e. the second phase is of primary importance. Furthermore, in [HCC 93] the strategy for attribute selection in the second phase is not specified. In practical applications, not only one but several threshold values *T* may be of interest. E.g., a value of 10 may be required for manual analysis by a manager, a value of 200 may be useful for analysis by some technical staff, and *T* = 10.000 may be appropriate to efficiently run a classification algorithm. Each of the *T* values requires an independent run of the generalization algorithm, since the selection of the next attribute depends on the value of *T* in the first phase, i.e. if $d_i < T$ then A_{i+1} is selected. E.g., the attribute sequences for $T = 10$ and $T = 20$ may be $[A_1^{\,l}]$ and $[A_1^{\,l-1}, A_2]$ resp. implying that the fully generalized relation for $T = 20$ cannot be used as input to create the fully generalized relation for $T = 10$.

These drawbacks are caused by the following features of the basic generalization algorithm: first, by the two phases of generalization and, second, by the depth first approach to attribute selection in the first phase. Therefore, we propose a modified gener-

```
FOR each attribute A_i DO
   calculate d_i;
   WHILE d_i > T DO
      IF C_i available THEN
         FOR each tuple of R DO
            replace the value of A_i by its predecessor in C_i;
         END FOR;
         Sort R, eliminate redundant tuples, update support;
         calculate d_i;
      ELSE
         eliminate A_i from R;
         Sort R, eliminate redundant tuples and update
         support;
      END IF;
   END WHILE;
END FOR first phase;
/* All attributes A_i have at most T different values in R */
WHILE card(R) > T DO
   Select next A_i according to some strategy;
   IF C_i available THEN
      FOR each tuple of R DO
         replace the value of A_i by its predecessor in C_i;
      END FOR;
   ELSE
      eliminate A_i from R;
   END IF;
   Sort R, eliminate redundant tuples and update support;
END WHILE second phase;
```

figure 1: **`Basic_Generalize(Relation R, Threshold T)`**

alization algorithm using a single phase and following a breadth first approach. In each iteration, the attribute A_i with the maximum d_i value is selected for generalization. If this criterion yields several attributes then from these the A_i with minimum g_i is chosen. This strategy of selection yields a fully generalized relation with similar numbers of attribute values at similar conceptual levels for all remaining attributes.

Our algorithm called *Homogeneous_Generalize* takes a list of T values sorted in descending order as argument and is presented in figure 2. *Homogeneous_Generalize* obtains the fully generalized relations for multiple values of T in one run which is the contents of lemma 1. Consequently, the generalization for multiple T values can be performed as efficient as for a single T value.

Lemma 1

Algorithm Homogeneous_Generalize is *correct* for multiple values of T , i.e. the fully generalized relations for each element t of *T_List* are identical with the relation obtained when performing Homogeneous_Generalize with a single value of $T = t$.

Proof Idea: The attribute selection of Homogeneous_Generalize is independent of the T values.

```
WHILE card(R) > LAST(T_List) DO
   Select A_i with maximum d_i;
   IF A_i not unique THEN
      Select from those A_i the one with minimum g_i;
      IF A_i not unique THEN
         Select first of those A_i in lexicographic order;
      END IF;
   END IF;
   IF C_i available THEN
      FOR each tuple of R DO
         replace the value of A_i by its father in C_i;
      END FOR;
      calculate d_i;
      increment g_i;
   ELSE
      eliminate A_i from R;
   END IF;
   Sort R, eliminate redundant tuples and update support;
   IF card(R) < NEXT(T_List) THEN
      store R as the result for T = NEXT(T_List)
   END IF;
END WHILE;
```

figure 2: `Homogeneous_Generalize(Relation R, Thresholds T_List)`

3. Requirements for Incremental Generalization

Updates of the base relation require updates of all derived relations. We call these updates of the generalized relations *incremental* if they do not require a new run of the generalization algorithm on the updated base relation but only on a generalized relation different from the base relation. In this section, we discuss the general requirements for an incremental generalization algorithm, i.e. efficiency, correctness and avoidance of overly generalization.

Clearly, the goal of incremental generalization is a significant gain in *effciency* vs. non-incremental generalization so that the derived relations can be updated during the time when the data warehouse is not used, e.g. during night.

We call an incremental generalization algorithm *correct* if the resulting fully generalized relation is identical to the result of applying the non-incremental generalization algorithm to the updated base relation.

The problem of overly generalization was informally introduced in [HCC 93] without presenting exact definitions and solutions. To define overly generalization, we introduce some notions.

Definition 6: overly generalized

A fully generalized relation R is called *overly generalized* w.r.t. some generalization algorithm, if its generalization level G_R is larger than the generalization level obtained when applying this generalization algorithm to the base relation of R.

Definition 7: unnecessary generalizations, overly generalization factor

Let G_{Rnew} be the generalization level of a fully generalized relation R obtained when applying a generalization algorithm to the updated base relation. Let G_{Rinc} be the generalization level of the fully generalized relation R obtained when applying the same generalization algorithm to an updated generalized relation different from the base relation. We define the *number of unnecessary generalizations* of R, denoted as NUG_R, as follows:

$$NUG_R = G_{Rinc} - G_{Rnew}$$

If $G_{Rnew} \neq 0$, then we define the *overly generalization factor* of R (denoted as OGF_R):

$$OGF_R = \frac{NUG_R}{G_{Rnew}} + 1$$

The overly generalization factor measures the relative number of unnecessary generalizations. If $NUG_R = 0$, then $OGF_R = 1$. The following very small example shows that, in general, after deletions from the base relation overly generalization cannot be avoided unless the generalization algorithm is applied to the whole updated base relation.

Example

We assume a base relation with attributes "sex" and "age" and the following tuples:

Sex	Age	Support
male	45	1
female	48	1
male	62	1

Let $T = 2$, $d_{Sex} = 2$, $d_{Age} = 3$. "Age" is the first attribute selected for generalization resulting in the following relation:

Sex	Age	Support
male	medium	1
female	medium	1
male	old	1

Now $d_{Sex} = d_{Age} = 2$ and $g_{Sex} < g_{Age} = 1$. "Sex" is the next attribute selected for generalization resulting in the fully generalized relation with $card(R) \leq T = 2$:

Sex	Age	Support
ANY	medium	2
ANY	old	1

Now, let the tuple (male,45,1) be deleted from the base relation.

A naive incremental algorithm proceeds as follows. First, the deleted tuple is generalized to the level of the fully generalized relation in order to determine which generalized tuple is afflicted by the deletion. In this example, it is the tuple (ANY, medium, 2). Second, the support of this tuple is decreased. Finally, the resulting relation is returned as the result of the incremental generalization:

Sex	Age	Support
ANY	medium	1
ANY	old	1

However, $NUG_R = 2$ for this relation since the updated base relation, depicted below, immediately fulfills the termination condition $card(R) \leq T = 2$:

Sex	Age	Support
female	48	1
male	62	1

Whereas in many data warehouses no deletions are performed, there are important cases where deletions are desirable:

- a data warehouse of an internet provider may contain a relation with the following schema keeping track of the accesses to the WWW pages of the provider:

Source URL	Destination URL	User	Day	Time
. . .	. . .	. . .	. . .	. . .

  Often, URL's are changed or deleted and analyses of the data warehouse should not consider these old URL's so that deletions will be performed for the tuples related to these URL's.
- imagine a protein database with a relation describing protein surface points with the following schema:

Protein Name	X	Y	Z	Electrostatic Potential	Hydrophobicity
. . .	.	. .	. .	. . .	. . .

  The protein data are obtained by complex analyses. The results of these analyses may be improved for a protein, e.g. using better equipment. Then, the tuples related to that protein have to be updated, i.e. they are first deleted and then reinserted with new attribute values.

Therefore, we have to deal with the problem of overly generalization. Clearly, if overly generalization occurs, the condition of correctness is violated as well. Thus, we have the following trade-off between correctness and overly generalization vs. efficiency:

- if we require strict correctness and no overly generalization then the generalization algorithm has to be applied to the updated base relation - which is very large - and is inefficient.

- if we do not care about overly generalization then we may run the generalization algorithm on the updated fully generalized relation - which is relatively small - and is very efficient.

In the following section, we present an approach to find a reasonable compromise between these conflicting requirements.

4. Incremental Generalization Algorithms

While the goal of efficiency requires to perform updates on the fully generalized relations, the goal of avoiding overly generalization requires to perform updates on the base relation. To find a reasonable compromise, we introduce the concept of anchor relations with a generalization level in between the base and the fully generalized relation (subsection 4.1). In subsections 4.2 and 4.3, incremental algorithms for insertions and deletions based on these anchor relations are presented.

4.1 Construction of the Anchor Relation

Our incremental approach to generalization is as follows:

1.) insertions and deletions are performed on the anchor relation

2.) the generalization algorithm is applied to the updated anchor relation.

Thus, the anchor relation is the relation used as input for incremental generalization. On one hand, the cardinality of the anchor relation should be significantly smaller than the cardinality of the base relation *R*. On the other hand, the cardinality of the anchor relation should be significantly larger than the cardinality of all fully generalized relations derived from *R*. Typically, the relevant thresholds *T* and, consequently, the cardinality of the fully generalized relations depend on the cardinality of the base relation, i.e. they are proportional to *card(R)*. Therefore, we specify the cardinality of the anchor relation as a certain fraction of the cardinality of the base relation. This fraction is a parameter (called anchor reduction factor) which can be set by the database administrator using his knowledge of the relevant generalization thresholds. In the following, we define the related notions.

Definition 8: anchor reduction factor, anchor threshold

Let *ARF* denote the *anchor reduction factor*. We define the *anchor threshold* for the base relation *R*, denoted by T_{Anc}, as

$$T_{Anc} = \frac{card(R)}{ARF}$$

The *anchor relation* of R, denoted as R_{Anc}, is the first generalization of *R* with $card(R_{Anc}) \leq T_{Anc}$.

4.2 Incremental Insertions

After insertions of tuples to the base relation *R*, a generalized relation derived from *R* may violate its threshold *T*, i.e. further generalization(s) are required. We assume that insertions are not performed tuple by tuple but that sets of tuples of considerable size are inserted at the same time. This approach is used in a data warehouse context: A data warehouse is not updated continously, but insertions and deletions of the base relations are collected and are applied to the data warehouse periodically, e.g. during the night. Then, all derived information, e.g. generalized relations derived from the warehouse, have to be updated as well before the operation of the warehouse starts again, e.g. in the next morning.

On insertion of a set of tuples to the base relation, these tuples are generalized to the generalization level of the anchor relation R_{Anc}. For all of these generalized tuples already present in R_{Anc}, the support of the existing tuple is incremented. If one of these generalized tuples is new in the anchor relation it is inserted with a support of 1. The generalization algorithm has to be applied to the updated anchor relation only if at least one new tuple was inserted to the anchor relation. In figure 3, the incremental insertion algorithm is presented.

```
FOR each t IN T_Set DO
   generalize t to the generalization level of R_Anc
      and store the resulting tuple in R';
end FOR;
Sort R', eliminate redundant tuples and update support;
new-start := FALSE;
FOR each t' IN R' DO
   IF NOT(t' IN R_Anc) THEN
      new-start := TRUE;
   END IF;
   insert t' into R_Anc;
END FOR;
Sort R_Anc, eliminate redundant tuples and update support;
IF new_start THEN
   Homogeneous_Generalize(R_Anc, Threshold_List);
ELSE
   FOR each t' IN R' DO
      FOR each threshold IN Threshold_List DO
         generalize t' to the generalization level of the fully
         generalized relation for threshold (R_thr) and increment
         support of the respective tuple in R_thr;
      END FOR each threshold;
   END FOR each t';
END IF new_start;
```

figure 3: Incremental_Insert(Relation R, Set_Of_Tuples T_Set)

4.3 Incremental Deletions

In section 3, we demonstrated that deletions of tuples from the base relation R may result in an overly generalization of a generalized relation derived from R which should be avoided as far as possible. As in the case of insertions, we assume that deletions are not performed tuple by tuple but that sets of tuples of considerable size are deleted at the same time.

On deletion of a set of tuples from the base relation, these tuples are generalized to the generalization level of the anchor relation R_{Anc}. For all of these generalized tuples, the support of the related tuple in R_{Anc} is decremented and if the support becomes 0 this tuple is deleted. The generalization algorithm has to be applied to the updated anchor relation only if at least one tuple was deleted from R_{Anc}. This application is due to the fact that a generalized relation derived from R_{Anc} may now have become overly generalized. Otherwise, if no tuple was deleted from R_{Anc} then no tuple will be deleted in a generalized relation derived from it since the support of tuples increases monotonically during generalization. Then, only the support of the related tuples in the fully generalized relations derived from R_{Anc} is decremented. In figure 4, the incremental deletion algorithm is presented:

```
FOR each t IN T_Set DO
   generalize t to the generalization level of R_Anc
      and store the resulting tuple in R';
end FOR;
Sort R', eliminate redundant tuples and update support;
new-start := FALSE;
FOR each t IN R_Anc DO
   decrement support of t by support of t IN R';
   IF remaining support of t = 0 THEN
      delete t from R_Anc;
      new-start := TRUE;
   END IF;
END FOR;
Sort R_Anc, eliminate redundant tuples and update support;
IF new_start THEN
   Homogeneous_Generalize(R_Anc, Threshold_List);
ELSE
   FOR each t' IN R' DO
      FOR each threshold IN Threshold_List DO
         generalize t' to the generalization level of the fully
         generalized relation for threshold (R_thr) and decrement
         support of the respective tuple in R_thr;
      END FOR each threshold;
   END FOR each t';
END IF new_start;
```

figure 4: Incremental_Delete(Relation R, Set_Of_Tuples T_Set)

5. Performance Evaluation

The proposed algorithms were implemented in C/Embedded SQL using ORACLE 7 on an HP 9000/735 workstation with HP-UX 9.01. In this section, we explain the design of the experiments (subsection 5.1) and present their results for incremental insertions (subsection 5.2) and deletions (subsection 5.3).

5.1 Design of the Experiments

We used real data from the protein database BIOWEPRO as test data. The goal of the BIOWEPRO project is to support protein-protein docking. A necessary condition for protein-protein docking is the complementarity of the interaction sites with respect to surface shape, electrostatic potential, hydrophobicity etc. The protein-protein docking database manages the 3D coordinates of the protein surface points as well as several geometric and physicochemical features for each point of the protein surface. The database is modeled as a relation with 9 attributes with a concept hierarchy each.

We chose a relation containing 112300 tuples, called the *source relation*, from which we derived several smaller relations by selecting a given percentage of all tuples. The tuples for the base relation were taken from the begin of the source relation and the tuples to be inserted were taken from the end. This was due to the fact that neighboring tuples in the relation are likely to represent neighboring points on the protein surface which have similar attribute values. Thus, the inserted tuples tend to have attribute values different from the existing tuples of the base relation implying a significant number of new tuples in the anchor relation.

Clearly, the performance results depend on a lot of parameters. We chose the following as *fixed parameters*:

- concept hierarchies
  The height of the concept hierarchies was 3 or 4.
- generalization threshold
  We set the generalization threshold to 50.

We chose the following as *variable parameters*: cardinality of the base relation, number of inserted tuples, number of deleted tuples and anchor reduction factor (*ARF*).

5.2 Results for Insertions

The efficiency of the incremental algorithms has to be compared with the efficiency of the non-incremental algorithm which is, therefore, evaluated first. Figure 5 presents the runtime of algorithm *Homogeneous_Generalize* depending on the cardinality of the base relation, i.e. *card(R)*. This figure confirms the expectation that the runtime is $O(n \log n)$ with $n = card(R)$ due to the dominating influence of the sorting step of the generalization algorithm [HCC 93]. Furthermore, figure 5 shows that the runtime is quite large even for relatively small relations.

The runtime of both, incremental insertions and deletions consists of two components:

- T_1 : runtime for generalizing the inserted / deleted tuples to the level of the anchor relation
- T_2 : runtime for updating (and possibly generalizing) the anchor relation

T_2 depends on the number of updates and on the cardinality of the anchor relation. Figure 6 shows the runtime for sets of incremental insertions on a base relation of 20.000 tuples depending on the number of new tuples for several *ARF's*. This runtime decreases significantly with increasing *ARF* (and decreasing cardinality of the anchor

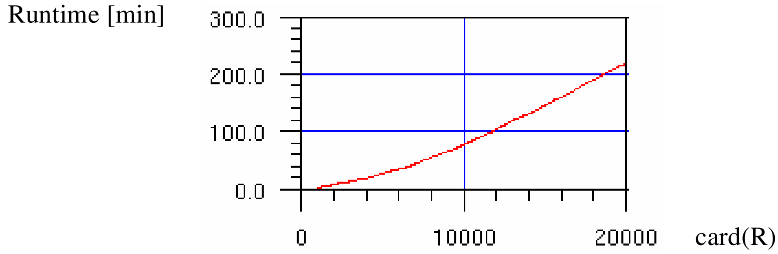

figure 5: Runtime of (non-incremental) algorithm Homogeneous_Generalize

relation) and increases superlinearly with increasing number of inserted tuples.

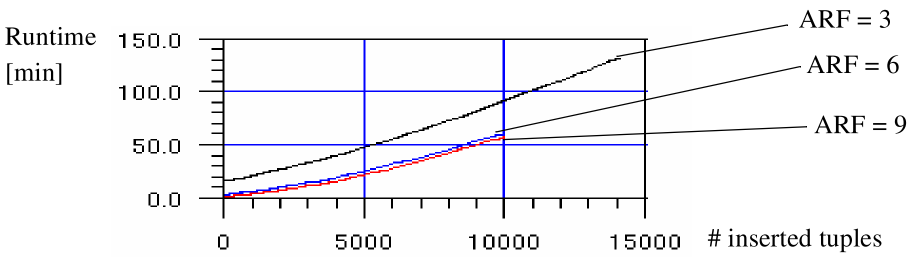

figure 6: Runtime of sets of incremental insertions for card(R) = 20000

In the following, we analyze the runtime for a single incremental insertion by calculating the average over all insertions. Figure 7 depicts the average runtime for one incremental insertion to a base relation of 20000 tuples depending on the total number of insertions with different anchor reduction factors.

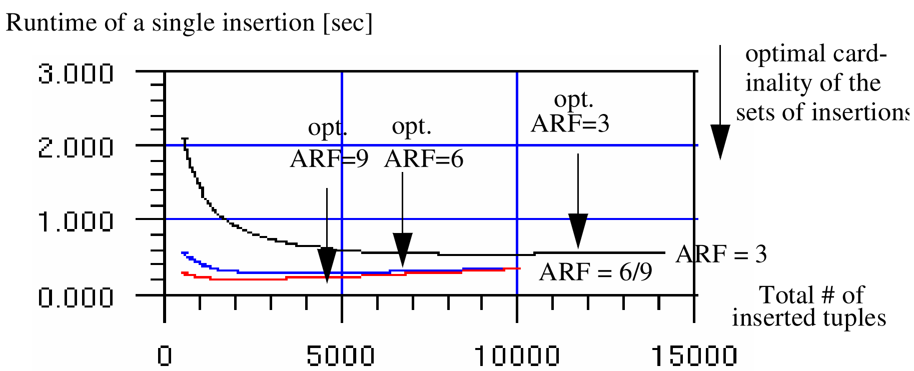

figure 7: Runtime of single incremental insertions for card(R) = 20000

We observe that the average runtime first decreases and then increases again. This is due to the following facts. The insertion of a single tuple (into the base relation) which creates a new tuple in the anchor relation causes the application of the generalization algorithm to the anchor relation. Therefore, the incremental insertion algorithm has a considerable minimum runtime which increases only slightly if several new tuples are created in the anchor relation. This minimum runtime can be regarded as an additive constant and its influence decreases with increasing size of the set of insertions because the ratio of the runtime (of the incremental insertion algorithm) and the number of inserted tuples decreases. On the other hand, the runtime of attribute-oriented generalization is *O(n log n)* for a relation of *n* tuples, i.e. superlinear, implying that the above ratio, i.e. the runtime of a single insertion, increases with increasing number of inserted tuples.

Due to these two conflicting trends, there is an *optimal cardinality* of the sets of insertions, i.e. a cardinality minimizing the ratio of of the runtime of the incremental insertion algorithm and the number of inserted tuples. Our experimental evaluation indicates that the minimum runtime and, consequently, the optimal cardinality increases with increasing cardinality of the anchor relation.

Thus, the set of all insertions should be split into subsets of the optimal cardinality to speed up the overall runtime. Figure 8 compares the runtime for insertions to a base relation *R* with *card(R)=5000* and *ARF=3* for the three different approaches:

- apply the generalization algorithm to the updated base relation (*non-incremental*)
- apply the incremental insertion algorithm to the whole set of insertions (*incremental-standard*)
- split the whole set of insertions into subsets with optimal cardinality and apply the incremental insertion algorithm to all of these subsets (*incremental-optimal*).

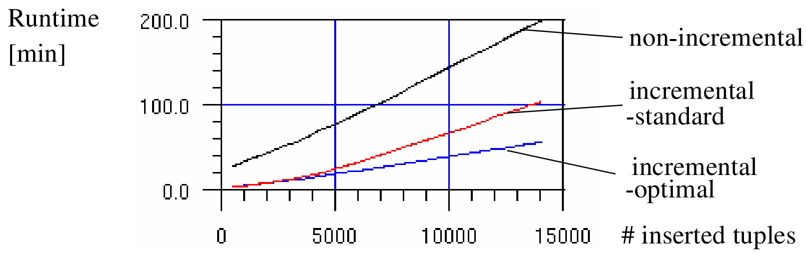

figure 8: Comparison of efficiency for the different strategies

Figure 8 demonstrates that the incremental approach is always significantly more efficient than the non-incremental one. Furthermore, inserting optimal sets of tuples reduces the runtime even more. And, finally, this gain in efficiency increases with increasing cardinality of the base relation and with increasing number of tuples to be inserted.

5.3 Results for Deletions

The runtimes for sets of incremental deletions are similar to those for insertions. Figure 9 depicts the runtime for sets of incremental deletions on a base relation of 20.000 tuples depending on the number of deleted tuples for several *ARF's*. This runtime decreases with increasing *ARF* and increases superlinearly with increasing number of deleted tuples.

In the following, we analyze the overly generalization factor (OGF) of the incremental deletions. We incrementally deleted 90 % of the tuples of the resp. base relations and calculated the resulting *OGF*. Figure 10 depicts the *OGF* after these incremental deletions depending on the *ARF* for different cardinalities of the base relation *R*.

We observe that the OGF increases with increasing ARF and decreasing cardinality

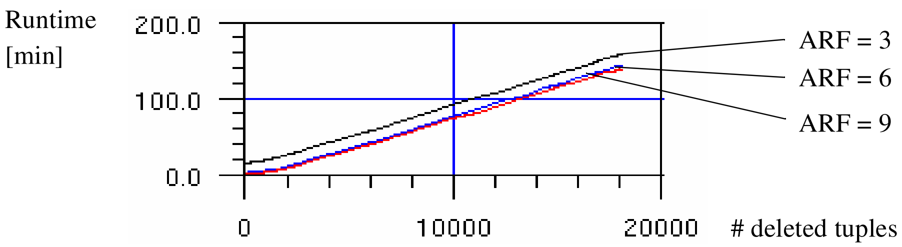

figure 9: Runtime of sets of incremental deletions for card(R) = 20000

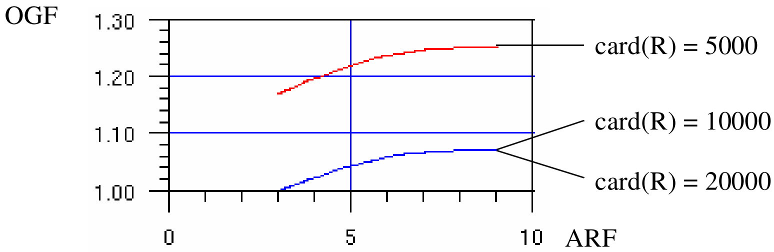

figure 10: Overly generalization factor after incremental deletions

of the base relation, i.e. the OGF increases with decreasing cardinality of the anchor relation. All values of the OGF, however, are moderate. Note that overly generalization only occurred when a very large percentage of all tuples was deleted from the base relation. We conclude that our approach for incremental generalization yields a good trade-off between efficiency (runtime) and effectivity (overly generalization) if the cardinality of the anchor relation is reasonably large.

6. Conclusions

Attribute-oriented generalization is a common method for the data mining task of summarization [HCC 93]. In this paper, we presented the algorithm Homogeneous_Generalize for attribute-oriented generalization with two important changes compared to the algorithm proposed in [HCC 93]: First, it has a single phase and strategy instead of two phases with different selection strategies. Second, it uses a breadth first approach instead of a depth first approach. The advantages of Homogeneous_Generalize are as follows: It obtains the fully generalized relations for multiple values of the generalization threshold in one run. Algorithm Homogeneous_Generalize is better suited for incremental updates since it obtains similar generalization levels for each attribute.

Typically, updates of an operational database are collected and applied to the data warehouse periodically. Then, all derived information, e.g. a generalized relation, has to be updated as well. We call updates of a generalized relation incremental if they do not require a new run of the generalization algorithm on the updated base relation. We discussed the requirements for incremental generalization, i.e. efficiency, correctness and avoidance of overly generalization. In particular, we showed that overly generalization may occur if deletions are performed on the base relation and introduced the overly generalization factor (OGF) as its measure. We presented algorithms for incremental insertions and deletions based on the materialization of a relation at an intermediate generalization level, i.e. the anchor relation. Our experiments demonstrate that incremental generalization can be performed efficiently with a low OGF.

Our experiments revealed the existence of an optimal cardinality for the sets of updates yielding a nearly linear runtime. A method for the automatic calculation of these optimal cardinalities and an algorithm to split the actual set of updates into subsets of (nearly) optimal cardinality should be developed. Furthermore, the performance of incremental generalization in other applications should be investigated.

References

[AS 94] Agrawal R., Srikant R.: *“Fast Algorithms for Mining Association Rules”*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, 1994, pp. 487-499.

[CHNW 96] Cheung D.W., Han J., Ng V.T., Wong Y.: *“Maintenance of Discovered Association Rules in Large Databases: An Incremental Technique”*, Proc. 12th Int. Conf. on Data Engineering, New Orleans, USA, 1996, pp. 106-114.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: “*A Database Interface for Clustering in Large Spatial Databases*”, Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, Montreal, Canada, 1995, AAAI Press, 1995, pp. 94-99.

[EKSX 96] Ester M., Kriegel H-P, Sander J. and Xu X.: *“A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”,* Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, 1996, pp. 226-231.

[FSS 96] Fayyad U., Piatetsky-Shapiro G., and Smyth P.: *“Knowledge Discovery and Data Mining: Towards a Unifying Framework”,* Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, 1996, pp.82-88.

[HCC 93] Han J., Cai Y., Cercone N.: *“Data-driven Discovery of Quantitative Rules in Relational Databases”*, IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.1, 1993, pp. 29-40.

[HF 96] Han J., Fu Y.: *“Exploration of the Power of Attribute-Oriented Induction in Data Mining”*, in *Advances in Knowledge Discovery and Data Mining,* AAAI/MIT Press, 1996, pp. 399-421.

[Huy 97] Huyn N.: *“Multiple-View Self-Maintenance in Data Warehousing Environments”*, Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, 1997, pp. 26-35.

[MQM 97] Mumick I.S., Quass D., Mumick B.S.: *“Maintenance of Data Cubes and Summary Tables in a Warehouse”*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 100-111.