

Optimizing Relational Store for E-Catalog Queries: a Data Mining Approach*

Min Wang

Data Management Department
IBM T. J. Watson Research Center
30 Saw Mill River Road
Hawthorne, NY 10532, USA
email: min@us.ibm.com

X. Sean Wang

Information and Software Eng. Department
George Mason University
4400 University Drive
Fairfax, Virginia 22030, USA
email: xywang@gmu.edu

Abstract

A frequent use of database management systems in electronic commerce is to provide electronic product catalogs (e-catalogs) that allow users to search for products of interest via constraints on attributes. An intuitively straightforward representation of e-catalogs is to use one table for the whole e-catalog as it is conceptually easy to maintain and query. However, for any e-commerce business with a reasonably large number of products and product types, its e-catalog usually involves a large number of attributes due to the great variety of the products, and at the same time, contains a large number of null values due to the fact that each product only has values under a relatively small number of attributes. Because of these properties, the above intuitive method does not work well in current relational database systems. Techniques have been proposed in the literature to deal with this problem, namely binary and vertical schemas. However, these techniques fail to take advantage of inherent properties of realistic e-catalogs to provide superior performance. This paper proposes a novel decomposition method for e-catalogs based on association rule discovery, a data mining technique. The method discovers groups of attributes that frequently appear together, i.e., are frequently used together to describe products, and generates schemas that contain these groups. This paper also reports experimental results showing the efficiency of the method.

Keywords: e-catalog, schema decomposition, association rule.

1. Introduction

Electronic commerce is emerging as a major application area for relational database management systems (RDBMS). One common use of RDBMS is for the support of electronic product catalogs (e-catalogs). The advantage of using RDBMS is

*The authors appreciate the comments of the anonymous reviewers. X. Wang also acknowledges the support by NSF under the career award 9875114.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to list, requires prior specific permission and/or a fee.

SAC 2002, Madrid, Spain
©2002 ACM 1-58113-445-2/02/03...\$5.00.

that it easily allows users to search for products of interest via constraints on attributes. However, due to special properties of e-catalogs, traditional use of RDBMS is not sufficient for situations where a large variety of products are present. Novel approaches are needed to deal with and take advantage of these special properties. This paper presents such a novel approach based on data mining techniques.

Consider an electronic marketplace for a large department store, such as Sears. The e-catalog of the marketplace may contain millions of products. Each product may have its own set of attributes to describe itself. For example, a particular “T-shirt” in woman’s apparel department may be associated with the attribute set $\{size, style, color, price\}$; and a particular “TV set” in electronic department may have a quite different attribute set $\{brand, view_type, signal_type, screen_size, price\}$. The total number of attributes across all the products can be huge.

If we store the information of all the e-catalog products in a relational database system, a conceptually easy way is to use the so-called *horizontal schema*. In this method, a product is represented as a row in a table, and the columns of the table are the union of all the attributes across all products.¹ This, however, results in a large number of columns in a table. Since the current commercial database systems support only a relatively small number of attributes per table (e.g., both DB2 and Oracle permit only up to 1012 columns in a table), using the horizontal schema becomes impractical.

Another problem of using the horizontal schema is due to the sparsity of the table. Indeed, compared to the overall number of attributes, most products in the e-catalog will only use a small number of attributes to describe. For example, any T-shirt product would have null values under all the attributes that are only relevant to TV sets, such as *view_type*. Not only the storage of these null values is wasteful, but more important, query performance will be poor using such a schema [ASX01], even if a RDBMS allowed huge number of columns.

A solution that has appeared in the literature is to use the so-called *binary schema* [CK85, KCJ⁺87]. Under this method, one table is created for each attribute, and it consists of two columns: one is the attribute column and the other is an *OID* column. The *OID* is an artificial value that identifies a product. The use of the *OID* column ties fields of a product across from all the relevant attribute tables.

The advantage of the binary schema is that it avoids the problem mentioned earlier for horizontal schema method. Indeed, the number of columns for all the tables is only two and thus commercial RDBMS can all handle them. Furthermore,

¹We will interchangeably use *column* (*row*, *table*, respectively) and *attribute* (*tuple*, *relation*, respectively) when the context is clear.

no null values need to be stored since if a product does not have a value for an attribute, then the corresponding table does not have to have a row to describe the product. However, even though there are no null values when using the binary schema, the number of join operations is usually large even when we process some not so complicated queries, hence the performance is still an issue.

As an alternative solution, some commercial e-commerce systems use the so-called *vertical schema*. Like the horizontal method, only one table is used. However, this table only has three attributes: *OID*, *Attr* (for attribute name), and *Value* (for attribute value).² Each product will be represented as a number of tuples in this table, one for each attribute under which the product has a value. Similar to the binary schema, multiple attributes of a product are tied together using the same *OID* across multiple tuples in the table. One problem with the vertical schema is that writing SQL queries against it becomes rather cumbersome. Recently, Agrawal et. al. proposed a method of creating a logical horizontal view on top of the vertical schema [ASX01]. However, the performance of query processing remains an issue when using vertical schema. As shown in [ASX01], vertical schema performs no better than binary schemas in most cases due to the large number of join operations involved in query processing.

In this paper, we introduce a method using a data mining technique. Our method is based on the following observation: Similar products will use similar group of attributes to describe, and there are many groups of similar products in the e-catalog. In this case, we may want combine the attributes in the group into one schema, instead of separating them into different tables (in the binary schema) or into different tuples (in the vertical schema).

There are two advantages if we combine attributes in a group into one table. First, there will not be many null values since a non-null value under one attribute usually means a non-null value under another in the group, and second, the chance that the attributes will be asked together in a query is greater than other groups of attributes since they are usually used together to describe a product.

Therefore, instead of binary decomposition, which ignores the structure of the e-catalog data, we try to do decomposition based on the “togetherness” of attributes. Intuitively, two attributes appear frequently together if a product has a (non-null) value under one attribute, then it has a (non-null) value under the other, and vice versa. This notion of togetherness can easily be extended to a group of attributes and is similar to the association rule notion that is used in data mining. An association rule is based on the concept of “large item” sets, which are sets of values that appear together frequently. In order to find groups of attributes that are frequently used together by products, we can naturally use association rule discovery techniques. More specifically, we can use association rule discovery method to discover attributes that frequently appear together, and use each such group of attributes as a schema.

Table 1 is an running example (in horizontal representation) that we will use throughout the rest of the paper. In the table, space represents a null value. The vertical and binary representations of Table 1 are shown in Figure 1. Note that the relation in Table 1 has many null values, or the density is low. As mentioned earlier, this is efficient in neither storage (storage wasted fro null values), nor processing time (each block will contain only a few non-null values).

The rest of the paper is organized as follows. In the next

²In reality, the *Value* column needs to be extended to accommodate values of different types.

<i>OID</i>	<i>A</i> ₁	<i>A</i> ₂	<i>A</i> ₃	<i>A</i> ₄	<i>A</i> ₅
1	<i>v</i> ₁	<i>v</i> ₂	<i>v</i> ₃		<i>v</i> ₄
2	<i>v</i> ₅	<i>v</i> ₆	<i>v</i> ₇		<i>v</i> ₈
3	<i>v</i> ₉	<i>v</i> ₁₀			<i>v</i> ₁₁
4			<i>v</i> ₁₂	<i>v</i> ₁₃	<i>v</i> ₁₄
5			<i>v</i> ₁₅		<i>v</i> ₁₆
6				<i>v</i> ₁₇	<i>v</i> ₁₈

Table 1: Horizontal Representation.

<i>OID</i>	<i>Attr</i>	<i>Value</i>
1	<i>A</i> ₁	<i>v</i> ₁
1	<i>A</i> ₂	<i>v</i> ₂
1	<i>A</i> ₃	<i>v</i> ₃
1	<i>A</i> ₅	<i>v</i> ₄
2	<i>A</i> ₁	<i>v</i> ₅
2	<i>A</i> ₂	<i>v</i> ₆
2	<i>A</i> ₃	<i>v</i> ₇
2	<i>A</i> ₅	<i>v</i> ₈
3	<i>A</i> ₁	<i>v</i> ₉
3	<i>A</i> ₂	<i>v</i> ₁₀
3	<i>A</i> ₅	<i>v</i> ₁₁
4	<i>A</i> ₃	<i>v</i> ₁₂
4	<i>A</i> ₄	<i>v</i> ₁₃
4	<i>A</i> ₅	<i>v</i> ₁₄
5	<i>A</i> ₃	<i>v</i> ₁₅
5	<i>A</i> ₅	<i>v</i> ₁₆
6	<i>A</i> ₄	<i>v</i> ₁₇
6	<i>A</i> ₅	<i>v</i> ₁₈

<i>OID</i>	<i>A</i> ₁
1	<i>v</i> ₁
2	<i>v</i> ₅
3	<i>v</i> ₉

<i>OID</i>	<i>A</i> ₂
1	<i>v</i> ₂
2	<i>v</i> ₆
3	<i>v</i> ₁₀

<i>OID</i>	<i>A</i> ₃
1	<i>v</i> ₃
2	<i>v</i> ₇
4	<i>v</i> ₁₂
5	<i>v</i> ₁₅

<i>OID</i>	<i>A</i> ₄
4	<i>v</i> ₁₃
6	<i>v</i> ₁₇

<i>OID</i>	<i>A</i> ₅
1	<i>v</i> ₄
2	<i>v</i> ₈
3	<i>v</i> ₁₁
4	<i>v</i> ₁₄
5	<i>v</i> ₁₆
6	<i>v</i> ₁₈

(a) Vertical representation (b) Binary representation

Figure 1: Vertical and binary representations.

section, we formally formulate our problem. In Section 3, we present our decomposition algorithm based on association rule discovery. We present our experiment results in Section 4 and conclude the paper in Section 5 with some pointers to future research directions.

2. Problem Formulation

We start by defining some basic notions used in the paper. An *e-commerce data schema* is a set of attribute names that contains attribute *OID*, which is a key. We will use the term *schema* to refer to e-commerce data schema when no confusion arises. An e-commerce data schema is similar to a relational schema with only the restriction that *OID* appears as a mandatory attribute and a key.

For example, the horizontal representation of Table 1 and the binary representation of Figure 1(b) are special cases of e-commerce data schemas.

In e-commerce applications, users usually search for desired products by providing bounds (constraints) on attribute values. For example, a user may be interested in finding all the T-shirts that satisfy all the following constraints: (a) *size* = 'M', (b) *color* = 'Red', and (c) $\$45 \leq \textit{price} \leq \50 . The most popular type of queries in e-catalog search has the following form if we use the horizontal schema:

```

SELECT OID
FROM E
WHERE (Ai1 not null) AND (bound on Ai1) AND
      (Ai2 not null) AND (bound on Ai2) AND
      ...
      (Aik not null) AND (bound on Aik)

```

In this paper, we only consider the above type of queries.

As mentioned in the introduction, our method calls for decomposition of the e-catalog data (which is conceptually represented in the horizontal schema) into e-commerce schemas.

Definition Let E be an e-commerce data schema. A *schema decomposition* of E is a set of e-commerce schemas $\{E_1, \dots, E_n\}$ such that $E = E_1 \cup \dots \cup E_n$.

For example, the set $\{\{OID, A_1, A_2\}, \{OID, A_3, A_5\}, \{OID, A_4\}\}$ is a decomposition of the schema in Table 1.

Definition Given a relation r on schema E , and a schema E_i such that $E_i \subseteq E$, the *null-tuple-free-projection* of r on E_i , denoted $\hat{\pi}_{E_i}(r)$, is defined as follows:

$$\hat{\pi}_{E_i}(r) = \{t[E_i] \mid t \in r \text{ and } t[E_i] \text{ contains at least two non-null values}\}.$$

Note in the above, since OID always appears in E_i and OID value is always not null, each tuple in $\hat{\pi}_{E_i}(r)$ must contain at least one non-null value other than the OID value. Also note that the null-tuple-free-projection is the same as a regular projection except that tuples with all null values (except its OID value) are eliminated.

A good decomposition of Table 1 is shown in Figure 2. In schema $\{OID, A_1, A_2\}$, the tuples with OID values 4, 5, and 6 (in Table 1) are eliminated since their projection contains only nulls under attributes A_1 and A_2 .

OID	A_3	A_5
1	v_3	v_4
2	v_7	v_8
3		v_{11}
4	v_{12}	v_{14}
5	v_{15}	v_{16}
6		v_{18}

OID	A_1	A_2
1	v_1	v_2
2	v_5	v_6
3	v_9	v_{10}

OID	A_4
4	v_{13}
6	v_{17}

Figure 2: A good decomposition of Table 1.

One goal of decomposition is to eliminate null values and hence increase the density of the resulting relations. The notion of density is defined in a usual way as follows.

Definition Given a relation r on schema E , the *density* of r is defined as the total number of non-null entries, other than the OID entries, in r divided by $|r| * (|E| - 1)$.

For example, the density of the horizontal table in Table 1 is 60%. Also, the density of the tables under the schemas $\{OID, A_1, A_2\}$ and $\{OID, A_4\}$ in Figure 2 both have density 1, while the table under the schema $\{OID, A_3, A_5\}$ has a density of $10 / (6 * (3 - 1)) = 83\%$.

The density of a decomposition can be defined by extending the above density definition as follows.

Definition Suppose the set of schemas $\{E_1, \dots, E_m\}$ is a schema decomposition of an e-commerce data schema E and r is a relation on schema E . The *overall density* of the schema decomposition is

$$density(E_1, \dots, E_m) = \sum_{1 \leq i \leq m} density(\hat{\pi}_{E_i}(r)) \times weight_i,$$

where

$$weight_i = \frac{|\hat{\pi}_{E_i}(r)| * (|E_i| - 1)}{\sum_{1 \leq j \leq m} |\hat{\pi}_{E_j}(r)| * (|E_j| - 1)}.$$

Informally, the density of a decomposition is the weighted sum of the density of all the relations. The weight of a relation is basically the ratio of the number of entries (null or non-null) of the relation versus the overall number of entries of all the relations involved. For example, the overall density of the schema decomposition in Figure 2 is about 81.7%. The overall density of the binary decomposition in Figure 1 is 100%.

The binary schema is formalized as follows.

Definition A *binary schema* is one with of the form $E_i = \{OID, A_i\}$, where A_i is an attribute. Given a schema $E = \{OID, A_1, \dots, A_n\}$, the *binary decomposition* of E is $E_1 = \{OID, A_1\}, \dots, E_n = \{OID, A_n\}$.

The overall size of binary decomposition tends to be (although not always) the smallest to answer a query, while the number of relations needed are the greatest. Using the horizontal relation, the number of relation needed to answer a query is the smallest (namely one), while the overall size tends to be the greatest (since it contains a lot of null values).

3. Schema Decomposition via Association Rule Discovery

In this section, we discuss the technique to decompose the horizontal relation into e-commerce schemas. As discussed in the introduction, the basic idea is to group attributes together based on the fact they almost always appear together to describe certain products while have all nulls when describing other products.

We use the association rule discovery [AIS93, AS94, SA96, HPY00, WHH00] to guide the schema decomposition.

3.1. Association rules and large item sets

The problem of discovering *association rules* was introduced in [AIS93]. Given a set of transactions, where each transaction is a set of items, an association rule is an expression of the form $X \Rightarrow Y$, where X and Y are sets of items. A popular example of an association rule that is often used in literature is: “40% of transactions that contain beer also contain diapers; 3% of transactions contain both beer and diaper.” Here 40% is called the *confidence* of the rule, and 3% the *support* of the rule. The problem is to find all association rules that satisfy user-specified minimum support and minimum confidence constraints.

When the association rule $X \Rightarrow Y$ satisfies the minimum support constraint, we also call the set $X \cup Y$ a “large item set”. By definition, a set Z of items is “large” if the following is true:

$$\frac{\text{The \# of times } Z \text{ appears as a subset in transactions}}{\text{Total \# of transactions}} \geq h,$$

where h is the support threshold.

3.2. Translating our problem to large item set discovery

As discussed in the introduction, intuitively, when two or more attributes in an e-commerce schema almost always have non-null or null values in unison, then it is usually beneficial to have these attributes appear in one of the decomposed schemas. This observation is formalized below.

Let r be a relation on the schema $E = \{A_1, \dots, A_n\}$. Define a candidate schema as a subset E_i of E such that in most of the tuples in r , either all E_i attributes are not null, or all E_i attributes are null. More precisely,

Definition Let E be an e-commerce schema and r a relation over E . Given a subset E_i of E , let $|\text{null}_{E_i}(r)|$ and $|\text{non-null}_{E_i}(r)|$ be the number of tuples in r whose E_i attributes are all null and all non-null, respectively. Then E_i is a *candidate schema* if

$$\frac{|\text{null}_{E_i}(r)| + |\text{non-null}_{E_i}(r)|}{|r|} \geq h,$$

where h is the given threshold value. The left hand-side of the above formula is called the *combined support* of the candidate schema E_i .

In the rest of the paper, we will use the term *support* to refer to combined support whenever it is clear in the context.

For example, in Table 1, in each tuple of the table, attributes A_1 and A_2 either both have non-null values or both have the null values. In other words, the schema $\{OID, A_1, A_2\}$ has the support of 1. As another example, the support of the schema $\{OID, A_3, A_5\}$ is $\frac{2}{3}$ in Table 1.

Definition A *maximum candidate schema* is one such that none of its super sets are candidate schemas.

The above definition of (maximum) candidate schemas can be translated into a variation of association rules studied in the literature [AIS93, AS94, SA96, HPY00, WHH00]. More specifically, we translate each tuple t in r into two subsets of E as follows: $\hat{E}(t) = \{A|A \in E \text{ and } t[A] \text{ is not null}\}$ and $E'(t) = \{A|A \in E \text{ and } t[A] \text{ is null}\}$. For example, take the tuple t with $OID = 3$ in Table 1. We have $\hat{E}(t) = \{A_1, A_2, A_5\}$ and $E'(t) = \{A_3, A_4\}$.

By extending the above definition, we see that the relation r gives rise to two bags of transactions, denoted $\text{non-nullTrans}(r)$ and $\text{nullTrans}(r)$, where $\text{non-nullTrans}(r)$ consists of all the transactions $\hat{E}(t)$ and $\text{nullTrans}(r)$ consists of all the transactions $E'(t)$, respectively, for all the tuples t in r . A subset E_i of E is a candidate schema precisely when the combined appearance of E_i in $\text{non-nullTrans}(r)$ and $\text{nullTrans}(r)$ is frequent. Here, the ‘‘combined frequency’’ is defined as follows:

$$\frac{|E_i(\text{non-nullTrans}(r))| + |E_i(\text{nullTrans}(r))|}{|r|},$$

where $|E_i(\text{non-nullTrans}(r))|$ is the number of times E_i appears in $\text{non-nullTrans}(r)$ and $|E_i(\text{nullTrans}(r))|$ is the number of times E_i appears in $\text{nullTrans}(r)$. Note that $|r|$ is the total number of tuples in r .

Association rule discovery algorithms appeared in the literature can be used to discover (maximum) candidate schemas. This is based on the following:

If

$$\frac{|E_i(\text{non-nullTrans}(r))| + |E_i(\text{nullTrans}(r))|}{|r|} \geq h$$

then

$$\begin{aligned} \text{either } & \frac{|E_i(\text{non-nullTrans}(r))|}{|\text{non-nullTrans}(r)|} \geq \frac{h}{2} \\ \text{or } & \frac{|E_i(\text{nullTrans}(r))|}{|\text{nullTrans}(r)|} \geq \frac{h}{2}, \end{aligned}$$

since $|\text{non-nullTrans}(r)| + |\text{nullTrans}(r)| = |r|$. Each of the above two formulas is exactly the formula used in the definition of large item sets E_i earlier in the section.

Following from the above discussion, our schema decomposition procedure can then be as follows.

(Step 1) Use the regular association rule discovery algorithm and half of the threshold to discover all the large item sets in $\text{non-nullTrans}(r)$ and $\text{nullTrans}(r)$, respectively.

(Step 2) Assume in Step 1, a large item set E_i is found in $\text{non-nullTrans}(r)$. Then count the number of times E_i appears in $\text{nullTrans}(r)$, and decide if the combined frequency exceeds the threshold. If yes, E_i is a candidate schema. Similarly for large item set E_i found in $\text{nullTrans}(r)$ in Step 1. This step will give us all the candidate schemas.

(Step 3) Find all maximum candidate schemas from the result of Step 2.

The resulting decomposition will use all the maximum candidate schemas of step 3.

We can easily see that the above procedure always produces a decomposition.

Proposition 1 *The above procedure always produces a decomposition of a horizontal schema, no matter what threshold value is used.*

To prove the above, we only need to show that each attribute appears in at least one of the maximum candidate schemas. This is easy to see since, for each threshold $0 \leq h \leq 1$, the schema $E_i = \{A_i\}$ is always a candidate. Indeed, because for each tuple t , A_i appears either in $\hat{E}(t)$ or in $E'(t)$, the total number of appearances of A_i in $\text{non-nullTrans}(r)$ and $\text{nullTrans}(r)$ equals to $|r|$.

4. Experimental Results

In this section, we evaluate the performance of our decomposition method through some experiments. We first describe our data generation process, and then demonstrate the effect of different (support) thresholds on the decomposition. We finally summarize what we learn from the experiments.

To study the performance characteristics of our method over a wide variety data distributions, we use our own synthetic data generator to generate the data sets and query sets.

To generate the data sets, we model the distribution of the non-null values in a horizontal table by the parameters defined in Table 2. The program will generate the sparse horizontal table H that consists of A columns plus the OID column. The number of tuples in the table is determined by the parameters C , CT_min , and CT_max . The non-null entries are mainly located in C clusters. The i th cluster, denoted cl_i ($1 \leq i \leq C$), covers CA_i attributes, where CA_i is a random number between CA_min and CA_max . The number of tuples that are covered by cluster cl_i is CT_i , a random number between CT_min and CT_max . We can easily see that the total number of tuples in the table is determined by the parameters C , CT_min , and CT_max , and it should fall into the range $[C \times CA_min, C \times CA_max]$. Also, the density of table H is in the range $[C * CA_min / |H|, C * CA_max / |H|]$.

In the experiment we report in the sequel, we use a data set generated with the following setup: The total number of attributes is 100 ($A = 100$), the number of non-null clusters is 4,000 ($C = 4000$), the number of attributes cover by one non-null clusters is between 5 and 10 ($CA_min = 5$ and $CA_max = 10$), and the number of tuples for each non-null cluster is between 100 and 300 ($CT_min = 100$ and $CT_max = 300$). This simulates the situation where we have 100-300 products that share the same attributes, while each product is described by 5-10 attributes, and we have 4,000 different kinds of products.

The generated horizontal table has 798,686 rows and 100 attributes. The density of the horizontal table is around 9.5%, i.e., among the approximately 80 million (79,868,600)

Parameter	Meaning
A	Total number of attributes
C	Number of non-null clusters
CA_{min}	Minimum number of attributes covered by one non-null cluster, $CA_{min} \geq 1$.
CA_{max}	Maximum number of attributes covered by one non-null cluster, $CA_{max} < A$
CT_{min}	Minimum number of tuples covered by one non-null cluster
CT_{max}	Maximum number of tuples covered by one non-null cluster

Table 2: Parameters for the synthetic data generator.

entries, only 9.5% of the values, or 758,746 values, are non-null. Each attribute value (including the *OID* and null value) is assumed to take up 4 bytes of storage. We further assume that the size of a disk block is 4,096 bytes. In the following, we always use number of disk blocks to measure the size of a table and the cost of a query.

We used minimum support thresholds 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 85%, 90%, 95%, 98%, and 100%. For the case of threshold 0%, we will get back the horizontal table since the whole set of attributes from the horizontal table would be a candidate schema by definition (it only needs to appear 0 times, i.e., no requirements at all). When minimum support threshold is 100%, we obtained the binary schema. This gives us a chance to compare our method with the binary schema method without specifically generating the binary schema. Note though, using threshold 100% does not necessarily return the binary schema.³

We show the results of the experiments in a series of figures. In Figure 3, we show the effect of the choices of minimum supports on the storage size, number of tables, average number of attributes per table, and overall density of the decompositions. The overall storage size (Figure 3(a)) goes beyond even the horizontal table (corresponding to minimum support 0%) when the minimum support requirement is low. This is due to the fact that some attributes appear in multiple tables in the decomposition. When the minimum support requirement becomes large enough, the storage requirement goes down, although attributes may still appear in multiple tables, due to the elimination of null values.

The number of tables (Figure 3(b)) goes up strictly because the smaller the minimum support requirement is, the fewer attributes in each table. Since the total number of attributes is a fixed number (100), we see an increase in the number of tables. At the same time, the average number of attributes per table decreases (Figure 3(c)). In Figure 3(d), the overall density of the decomposition goes up as expected. It should be observed that the trend is faster than linear.

Once we generated decompositions, we use the same data generation algorithm to generate clusters that simulate queries. That is, now each tuple is interpreted as a query, where a non-null value under an attribute is interpreted as the query having a constraint on the attribute. We generated two sets of queries. All the queries in query set 1 have 1-3 constraints (i.e., clusters with only 1-3 attributes having non-null values), and all the queries in query set 2 have 4-6 constraints. Query set 1 has 400 queries overall, and query set 2 has 500 queries overall. The number of constraints in our queries is small (in the range of 1-6) since people usually only use a small number of constraints to query e-catalogs [SA00].

Figures 4(a) and (b) show the average costs of the queries on the decompositions of different minimum support requirements. We measure the cost in terms of the number of blocks accessed to answer a query. To simplify our discussion, we assume that no indices are used, and if join is needed (when a query involves two or more tables), nested-loop join is used.

³Indeed, as an example, if there were no null values in the horizontal table, then the horizontal table itself is the decomposition satisfying 100% support ratio.

The reason that the cost goes down generally before minimum support requirements are before 80%-90% is due to the fact that tables become much more dense and smaller (in size) even though some joins are required. The jumps towards the end of the curve is due to the fact that the benefit we gain from higher density cannot compensate the cost of more joins, because in these cases, the cost of join takes control over all other costs.

Figures 4(c) and (d) show more details of Figures 4(a) and (b), respectively, around minimum support requirements 80%-95% as this is the area we have shown the most benefit of our method.

For our e-catalog, the above figures show that for query set 1, the best decomposition is to use minimum support rate of 90%, while for query set 2, the best is 80%.

We did not discuss the details about association rule mining algorithm in this section as any standard algorithm can be used for our purpose as explained in Section 3.

5. Conclusion

In this paper, we investigated the use of togetherness of attributes in e-catalogs to obtain better decomposition of tables. We provided algorithms to generate such decompositions and studied performance via experiments. The benefit of using our method shows clearly from the experiment data.

There is much to be done along this line of research. First of all, how to maintain the decomposition when products are inserted and deleted from the e-catalog. This involves some incremental update to the decomposition. One strategy of course is to run the decomposition algorithm given in the paper periodically. But this reorganization may be costly, and some on-line incremental methods are usually preferred. The second and related issue is that when users query the e-catalog, they should not ask directly on the decomposition. Instead, the conceptual horizontal table should be the target of their queries. A middleware is needed to handle this automatically. Unlike the vertical schema, this is not too difficult a task.

In this paper, we did not discuss the effect of the workload, i.e., the set of queries that are usually asked. When workload is known, decomposition should take the workload into consideration. It is expected that the benefit of our decomposition method will be even greater. As future work, we will incorporate the query workload into our decomposition algorithm.

Other issues that arise from our research include the consideration of indices as well as overlapping ratio amongst the tables in the schema decomposition. We plan to investigate these in our future work.

In this paper, we used association rules. Other techniques may also apply. For example, fuzzy functional dependencies [IM00] may be used to describe the “associations” among attributes. Rather than relational store, other data storage types such as like semi structured data store [ASB99] may also be useful for storing e-catalogs. How the association rules or other techniques can be used to optimize the storage and access is an interesting future research topic.

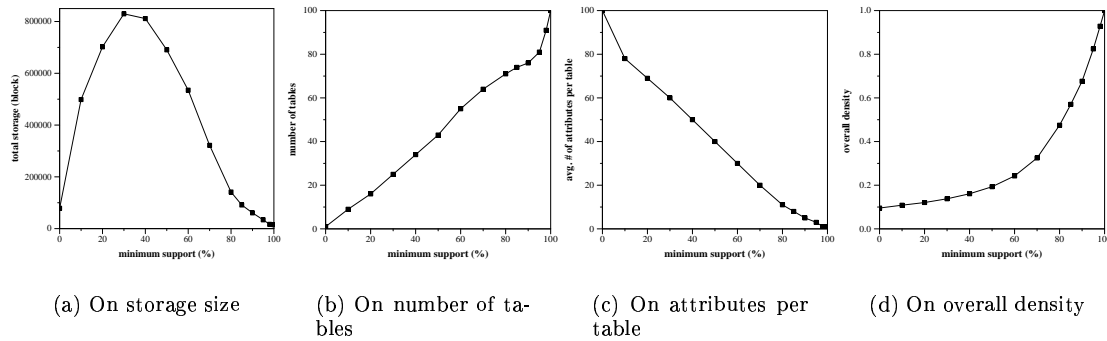


Figure 3: The effect of minimum support

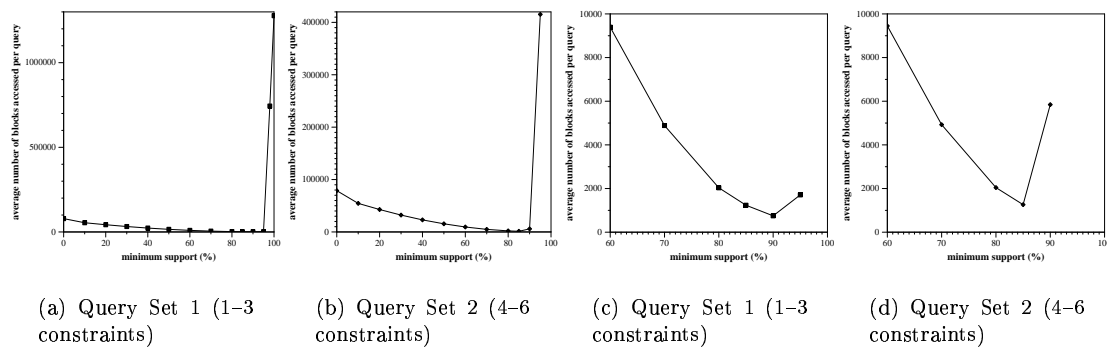


Figure 4: Average query cost vs. minimum support

References

- [AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 207–216. ACM Press, 1993.
- [AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB’94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann, 1994.
- [ASB99] Serge Abiteboul, Dan Suciu, and Peter Buneman. *Data on the Web : From Relations to Semistructured Data and Xml*. Morgan Kaufmann Publishers, 1999.
- [ASX01] Rakesh Agrawal, Amit Somani, and Yirong Xu. Storage and querying of e-commerce data. In *VLDB’01, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*. Morgan Kaufmann, 2001.
- [CK85] George P. Copeland and Setrag Khoshafian. A decomposition storage model. In Shamkant B. Navathe, editor, *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, Austin, Texas, May 28-31, 1985*, pages 268–279. ACM Press, 1985.
- [HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, pages 1–12. ACM, 2000.
- [IM00] Rolly Intan and Masao Mukaidono. Fuzzy functional dependency and its application to approximate data querying. In *Proc. International Database Engineering and Applications Symposium*, pages 47–54, 2000.
- [KCJ⁺87] Setrag Khoshafian, George P. Copeland, Thomas Jagodis, Haran Boral, and Patrick Valduriez. A query processing strategy for the decomposed storage model. In *Proceedings of the Third International Conference on Data Engineering, February 3-5, 1987, Los Angeles, California, USA*, pages 636–643. IEEE Computer Society, 1987.
- [SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining quantitative association rules in large relational tables. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 1–12. ACM Press, 1996.
- [SA00] John C. Shafer and Rakesh Agrawal. Continuous querying in database-centric web applications. In *Proceedings of the Ninth International World Wide Web Conference, May 15-19, 2000, Amsterdam*, pages 519–531, 2000.
- [WHH00] Ke Wang, Yu He, and Jiawei Han. Mining frequent itemsets using support constraints. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 43–52. Morgan Kaufmann, 2000.