Group Bitmap Index: A Structure for Association Rules Retrieval

Tadeusz Morzy, Maciej Zakrzewicz

Institute of Computing Science
Poznan University of Technology
ul. Piotrowo 3a, 60-695 Poznan, Poland
morzy@put.poznan.pl, mzakrz@cs.put.poznan.pl

Abstract

Discovery of association rules from large databases of item sets is an important data mining problem. Association rules are usually stored in relational databases for future use in decision support systems. In this paper, the problem of association rules retrieval and item sets retrieval is recognized as the subset search problem in relational databases. The subset search is not well supported by SQL query language and traditional database indexing techniques.

We introduce a new index structure, called Group Bitmap Index, and compare its performance with traditional indexing methods: B^+ tree and bitmap indexes. We show experimentally that proposed index enables faster subset search and significantly outperforms traditional indexing methods.

Introduction

Knowledge discovery in databases (KDD) is a new area of database research that aims at finding previously unknown and potentially useful patterns in large databases (Fayyad, Piatetsky-Shapiro, and Smyth 1996). The most commonly sought patterns are *association rules* (Agrawal, Imielinski, and Swami 1993), (Piatetsky-Shapiro and Frawley 1991), (Agrawal and Srikant 1994), (Savasere, Omiecinski, and Navathe 1995), (Toivonen 1996). Formally, by an association rule we mean a formula of the form $X \rightarrow Y$, where X and Y are two sets of items. Association rules are discovered from database tables that store sets of items. Consider a supermarket database where the set of items purchased by a customer on a single visit to a store is recorded as a *transaction*. The supermarket managers might be interested in finding *associations* among the items purchased together in one transaction. An example of a supermarket database and a set of association rules derived from the database are presented in Fig. 1. The example discovered rule: bread $\wedge$ butter $\wedge$ milk $\rightarrow$ apples states that a customer who purchases bread, butter and milk, probably also purchases apples. We refer to the left-hand side of the rule as *body*, and to the right-hand side as *head*. We also say, that the rule is *satisfied* by a given item set (item set *satisfies* the rule) if $X \cup Y$ is contained in the set. We say, that the rule is *violated* by a given item set (item set *violates* the rule) if the set contains X, but does not contain Y. Each rule has two measures of its statistical importance and strength: *support* and *confidence*. The support of the rule is the number of item sets that satisfy the rule divided by the number of all item sets. The rule confidence is the number of item sets that satisfy the rule divided by the number of item sets that contain X.

transaction_id	items
1	bread, butter
2	bread, butter, milk, apples
3	bread, butter, milk, apples

bread $\rightarrow$ butter

bread $\wedge$ butter $\wedge$ milk $\rightarrow$ apples

Fig 1. Example of a database and discovered rules

Usually, the discovered rules are stored in a database for future retrieval by users or decision support systems. The users may iteratively penetrate the set of rules discovered from the given database from many points of view. Moreover, users might be interested in finding customer transactions that e.g. violate a given association rule. Such queries, often referred to as relational division (Graefe and Cole 1995), require analyzing of set-valued attributes and are not supported by popular DBMSs.

In this paper we consider retrieval of association rules and item sets from a relational database. We generalize both retrieval problems to the subset search problem and introduce a new index structure, called *group bitmap index*. We show the results of the experiment in which the group bitmap index significantly outperforms the traditional indexing methods, namely B^+ tree and Bitmap indexes.

Storage Structures

The example data from the Fig.1 can be stored in a database table with two attributes: one referring to item values and one organizing items into sets or transactions. Each item is stored in a separate record and the item set may consist of many records. Such structure allows

efficient storage of variable length sets of items. An example purchase data table is depicted in Fig 2.

shopping

transaction_id	item
1	bread
1	butter
2	bread
2	butter
2	milk
2	apples
3	bread
3	butter
3	milk
3	apples

Fig. 2 Table for storing supermarket purchase data

Association rules that are mined in a knowledge discovery process can be also stored in database tables. Fig. 3 presents an example relational representation for association rules storage. The rule bodies and heads are placed in a separate table, while the second table keeps specific rule parameters (e.g. support and confidence). In this example, two rules from the Fig. 1 are represented.

rules

rule_id	supp.	conf.
1	0.83	0.90
2	0.25	0.13

elements

rule_id	item	type
1	bread	body
1	butter	head
2	bread	body
2	butter	body
2	milk	body
2	apples	head

Fig. 3 Example tables for rule storage

Notice the similarity between the representation of data item sets and the representation of rule items. Both representations are based on storage of specific items together with an identifier of the set the items belong to. For the sake of simplicity, we will assume further in the paper that both items and set identifiers are positive integer numbers.

Queries

We distinguish two basic types of queries that are usually issued in mining and searching of association rules:

A. Retrieve all item sets that contain a given subset of items (to determine the sets that satisfy or violate the specified rules).

B. Retrieve all rules that contain given subset of items in their bodies or heads.

Notice, that both types of the queries consist in finding the sets of items that contain a given item subset. A similar problem was studied in (Graefe and Cole 1995), where relational division operator was described. We will refer to this type of query as a *subset search* query. It is a set-oriented query that is not well supported by SQL interface and traditional database accessing methods. SQL language does not contain a subset search (or relational division) clause, therefore, to specify a subset search query in SQL, aggregation or multiple join clauses are required. Traditional accessing methods (B^+ tree, bitmap index, etc.) are row-oriented, i.e. they reference single records. Therefore, subset selection requires multiple use of the index. To illustrate the subset search query, we present below two examples of an SQL queries retrieving from a database table *data_table* the identifiers of data item sets containing four given items 0, 7, 12, and 13.

1.
```
select a.group_id
from data_table a,  data_table b,
     data_table c,  data_table d
where a.group_id = b.group_id
and b.group_id = c.group_id
and c.group_id = d.group_id
and a.item = 0 and b.item = 7
and c.item = 12 and d.item = 13
```

2.
```
select group_id  from data_table
where item in (0, 7, 12, 13)
group by group_id
having count(*) = 4
```

Table: **data_table**

group_id	item
1	0
1	7
1	12
1	13
2	2
2	4
3	10
3	17
3	20

In general (and we will show it experimentally), finding data item sets that contain a given subset is a complex and time-consuming task. Therefore, to provide faster subset searching we propose a new indexing technique.

Group Bitmap Index

In this Section we explain the idea of group bitmap indexing. The aim of the group bitmap indexing is to enable faster subset search and content-based association rules retrieval in relational databases. The key idea of the group bitmap index is to build a binary key, called *group bitmap key*, associated with each item set. The group bitmap key represents a content of the item set. During retrieval, the group bitmap keys are used to prune those sets of items that do not contain the searched subset.

In the following subsections we will introduce two types of group bitmap indexes. The first one, called *simple group bitmap index* will help us in explaining the basic ideas of the group bitmap index construction and utilization. Due to its properties the simple group bitmap index has rather theoretical character. Therefore, in the next subsection, we will present a modified index structure, called *hash group bitmap index*, that efficiently supports subset search queries and may be easily implemented in practice.

Simple Group Bitmap Index

The *simple group bitmap index* consists of a set of simple group bitmap keys. The *simple group bitmap key* for a set of items is a binary number, in which the bit value '1' on the position *k* indicates that the set contains the value *k*. The simple group bitmap keys are stored in an index table together with identifiers of the sets they refer to. When a

subset search query is issued, a simple group bitmap key is also computed for the searched subset of items. Then, the subset containment is checked by means of a fast bitwise machine operation, i.e. bitwise AND. The checking procedure consists in testing, if for every bit set to '1' in the simple group bitmap key of the searched subset, the corresponding bit of a simple group bitmap key of an item set is also set to '1'. The item sets whose simple group bitmap keys satisfy the testing condition are returned as the result of the subset searching. Figure 4 illustrates an example database table and the simple group bitmap index construction. Three simple group bitmap keys were derived for item sets stored in the database table. The derived simple group bitmap keys, together with corresponding item set identifiers, are stored in the index table.

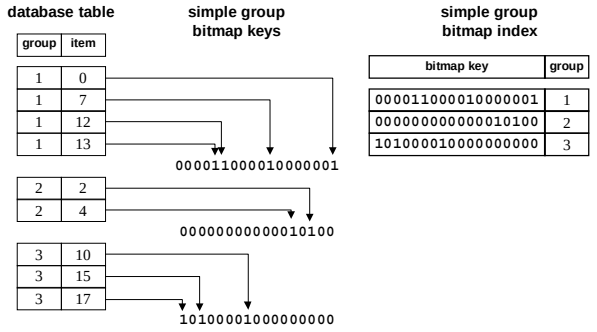

Fig. 4 Simple group bitmap index

When a subset search query seeking for item sets containing e.g. items 15 and 17 is issued, then the simple group bitmap key for the searched subset is computed. This key contains '1's on positions 15 and 17. In the next step, by means of a bitwise AND, the index table is scanned for keys containing '1's on the same positions. As the result, the item set identified by *group*=3 is returned.

Let us notice that the simple group bitmap keys have to be *N*-bit long, where *N* denotes the number of all possible items. In practice, *N* can be of order of hundreds or thousands. It results in very long, space-consuming simple group bitmap keys that are difficult to store as well as to process. Moreover, since the database is dynamic in nature that means that the number of possible items may change in time, then the length of simple group bitmap keys should change respectively. The maintenance of such index would be costly and difficult. Therefore, the simple group bitmap index has rather theoretical character. However, in case of a static database with a small number of possible items it could be applicable.

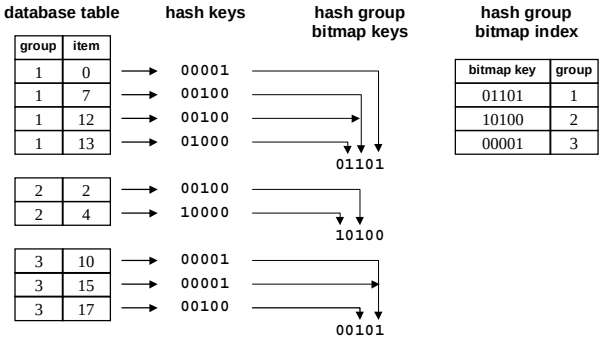

Fig. 5 Hash group bitmap indexing

Hash Group Bitmap Index

To eliminate the discussed disadvantages of the simple group bitmap index, we introduce the *hash group bitmap index*. This type of group bitmap index operates on hash group bitmap keys, whose length is *n*, where *n*<<*N*. The *hash group bitmap key* of an item set is created from the hash keys of all data items contained in a given item set, by means of bitwise OR operation. The *hash key* of the item X is an *n*-bit binary number defined as follows:

$$\texttt{hash\_key(X)} = 2^{(\texttt{X mod n})}$$

The subset search with hash group bitmap index is performed as the two-step procedure. The first step, called *filtering step*, consists in scanning the index and finding the identifiers of item sets that possibly contain the searched subset. It is done as follows. When a subset search query is issued, a hash group bitmap key is computed for the searched subset of items. Then, the hash group bitmap index is scanned and each hash group bitmap key is checked against the searched subset key. The checking procedure is performed by means of the bitwise AND machine operation, according to the following pseudo code:

X := hash_group_bitmap_key(searched_subset);
for each (*bitmap_key*, *group*) from *index table* do
if (*X* AND *bitmap_key* = *X*) then return *group*;

The identifiers of the item sets whose hash group bitmap keys satisfy the testing condition are returned as the result of the first step of the subset search procedure. However, due to the fact that hash keys do not uniquely represent items, the result of this step will possibly contain also false sets of items, i.e. the sets that in fact do not contain the searched subset. Therefore, the verification of the obtained result is necessary. This is the aim of the second step, called *verification step*. It simply consists in traditional selecting from the item sets found in the previous step, those item sets that contain the searched subset.

To illustrate the construction of a hash group bitmap

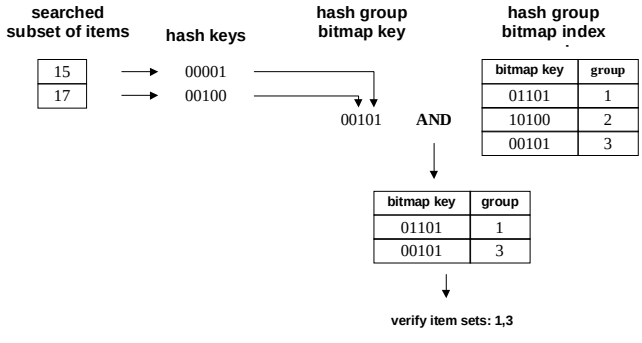

Fig. 6 Item sets retrieval with hash group bitmap index

index and its application to the subset search problem, let us consider the examples presented in Fig. 5 and Fig. 6. In the example in Fig. 5, three hash group bitmap keys were derived for item sets stored in the database table. When a subset search query seeking for item sets containing e.g. items 15 and 17 is issued, then the hash group bitmap key for the searched subset is computed (Fig. 6). Then, by means of a bitwise AND, the index table is scanned for keys containing '1's on the same positions. As the result of the first step of the subset search procedure, the item sets identified by *group*=1 and *group*=3 are returned. Then, in the verification step, these item sets are tested for the containment of the items 15 and 17. Finally, the item set identified by *group*=3 is the result of the subset search.

Experimental Results

The hash group bitmap indexing has been implemented on top of an *Oracle 7.3.2* RDBMS (Sun SPARCserver 630MP, 128 MB RAM). Experimental data sets were created by *GEN* generator from Quest project (Agrawal et al. 1996). Several parameters shown in Table 1 affect the distribution of the synthetic data.

parameter	**value**
n_{trans}	number of item sets, 50,000
n_{items}	number of different items, 100 to 500
t_{len}	average items per set, 15 to 30
n_{pats}	number of patterns, 500 and 10000
patlen	average length of maximal pattern, 4
corr	correlation between patterns, 0.25

Table 1 Synthetic data parameters

Fig. 7 shows the performance of traditional and hash group bitmap indexing methods for different sizes of a searched subset. Traditional B^+ tree and bitmap indexes show a rapid increase of query execution time for increasing searched subset size. The results of the search with hash group bitmap index do not significantly depend on the size of the searched subset. The cross-over between the hash group bitmap indexing and traditional bitmap indexing (which appeared the best of the traditional methods) occurs when a subset of 5 (for 24-bit group bitmap index) or 6 items (for 16-bit group bitmap index) is searched. For the searched subset size of 10 items, the hash group bitmap index allows data retrieval twice as fast as the traditional bitmap index.

The above experiment also showed that the best of

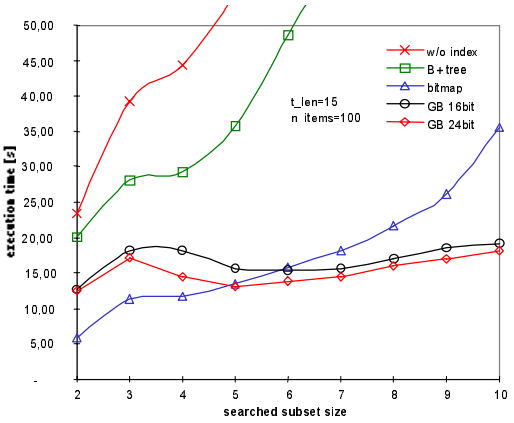

Fig. 7 Query execution time vs. searched subset size

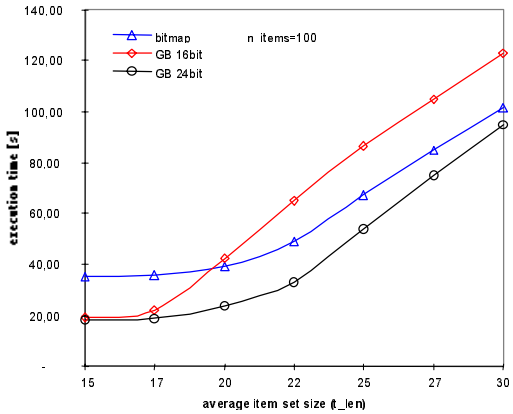

Fig. 8 Query execution time vs. average item set size

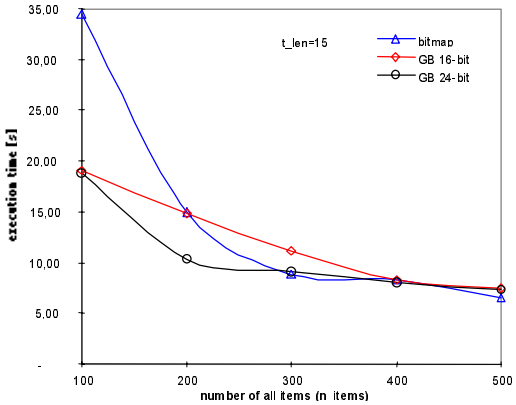

Fig. 9 Query execution time vs. number of items

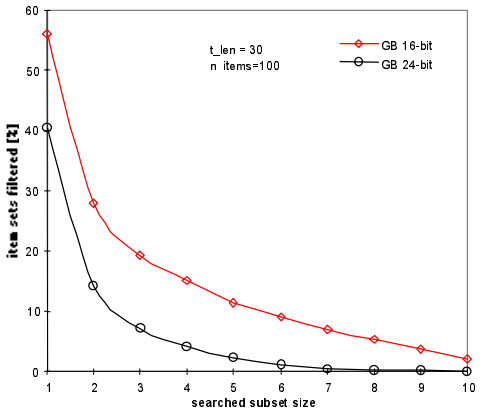

Fig. 10 Effectiveness of the filtering step

traditional database accessing methods is the one with the bitmap index. Therefore, in the subsequent experiments, we restrict our attention to the bitmap index, compared to our hash group bitmap index.

Fig. 8 demonstrates the effect of the average size of item sets on the performance of the hash group bitmap index and the bitmap index. As it was expected, the larger the average size of the sets, the longer hash group bitmap key should be used. When the average size is greater than the hash group bitmap key length, the hash group bitmap index does not improve the query execution. The conclusion is that the hash group bitmap key length should be slightly greater than the average item set size.

Fig. 9 demonstrates the effect of the number of items stored in the database on the performance of the hash group bitmap index and the bitmap index. For increasing number of items, the performance of the hash group bitmap index is getting relatively worse, compared to the traditional bitmap index. The explanation of this behavior is that more items are hashed to the same bits in the hash group bitmap key, thus increasing the number of false sets returned after the first step of the retrieval procedure. The false sets are removed from the result during the verification step, which therefore consumes more time. The conclusion is that the hash group bitmap key length should be greater for data sets having more items.

Fig. 10 illustrates the effectiveness of the filtering step of the subset search procedure. This effectiveness is measured by the percentage of item sets pruned at this step. As it can be seen, with the increase of the size of the searched subset, the percentage of pruned sets increases. For example, for the searched subset of the size 4-10, and 24-bit hash group bitmap index used, over 95% of all item sets are pruned. Besides, the difference between the effectiveness of 16-bit and 24-bit hash group bitmap index is demonstrated. For the same data set, the 24-bit index provides significantly better pruning than the 16-bit index. The conclusion is that pruning is performed better by hash group bitmap indexes with longer hash group bitmap keys.

The explanation of this behavior results from the idea of the hash group bitmap index. It can be shown that the number of bits set to '1' in *n*-bit hash group bitmap key for an item set of the size *L* is given by the following formula:

$$\overline{L} = \sum_{k=1}^{L} k \binom{n}{k} \frac{k^L + \sum_{i=1}^{k-1} (-1)^{k-i} \binom{k}{i} i^L}{n^L}$$

where: *n* is the number of bits of the hash group bitmap key, *L* is the number of items represented by the hash group bitmap key, $\overline{L}$ is the number of bits that are set to '1'. With increasing *L*, for the constant *n*, the number of items hashed to the same bits increase (e.g. for $L = 4$, $\overline{L} = 4$, while for $L = 16$, $\overline{L} = 10$, what means that six of ten bits have to represent two different items each). Similar behavior could be also observed for increasing the size of the searched subset. So, it is clear that for n=16 selectivity of our hash group bitmap key is less that for the hash group bitmap key of n=24.

Conclusions and Future Work

In this paper we introduced the new index type, called group bitmap index, that significantly reduce time of subset searching in large databases. This kind of searching has many applications in the field of data mining and association rules discovery. However, the new index may be also applied in traditional database systems to speed-up the execution of queries seeking for a subset of data items. We showed experimentally that the group bitmap index significantly outperforms traditional indexing methods including B^+ tree and bitmap indexing. The experiment was led on top of a DBMS, using standard SQL interface, therefore we believe that the results would be even better for the group bitmap index integrated into the core of DMBS.

References

Agrawal, R.; Imielinski, T.; Swami, A. 1993. Mining Association Rules Between Sets of Items in Large Databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC.

Agrawal, R.; Srikant, R. 1994. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th VLDB Conference, Santiago, Chile.

Fayyad, U.; Piatetsky-Shapiro, G.; Smyth, P. 1996. The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Comm. of the ACM, Vol. 39, No. 11.*

Imielinski, T.; Manilla, H. 1996. A Database Perspective on Knowledge Discovery. *Communications of the ACM, Vol. 39, No. 11.*

Piatetsky-Shapiro, G.; Frawley, W.J. editors 1991. *Knowledge Discovery in Databases*: MIT Press.

Savasere, A.; Omiecinski, E.; Navathe, S. 1995. An Efficient Algorithm for Mining Association Rules in Large Databases. In Proceedings of the 21st VLDB Conference, Zurich, Swizerland.

Toivonnen, H. 1996. Sampling Large Databases for Association Rules. In Proceedings of the 22nd VLDB Conference, India.

Graefe, G.; Cole, R.L. 1995. Fast Algorithms for Universal Quantification in Large Databases. *ACM Transactions on Database Systems, Vol. 20, No. 2.*

Agrawal, R.; Mehta, M.; Shafer, J.; Srikant, R.; Arning, A.; Bollinger, T. 1996. The Quest Data Mining System. In Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining, Portland, Oregon.