SET-ORIENTED INDEXES FOR DATA MINING QUERIES

Key words: data mining, subset search, bitmap index, hash index

Abstract: One of the most popular data mining methods is frequent itemset and association rule discovery. Mined patterns are usually stored in a relational database for future use. Analyzing discovered patterns requires excessive subset search querying in large amount of database tuples. Indexes available in relational database systems are not well suited for this class of queries. In this paper we study the performance of four different indexing techniques that aim at speeding up data mining queries, particularly improving set inclusion queries in relational databases. We investigate the performance of those indexes under varying factors including the size of the database, the size of the query, the selectivity of the query, etc. Our experiments show significant improvements over traditional database access methods using standard B+ tree indexes.

1 INTRODUCTION

Data mining, also referred to as knowledge discovery in databases, is a process of discovering novel, useful, valid and understandable patterns from large volumes of data (Fayyad et al., 1996). One of the most frequently used data mining techniques is the discovery of association rules. Informally, an association rule is an implication of the form X→Y where X and Y are disjoint sets of items. An example of an association rule might be *'milk'* ∨ *'butter'* → *'cereals'* representing the correlation of sales of products *'milk'*, *'butter'* and *'cereals'*. In other words, this rule represents the fact that a customer who buys both milk and butter is highly likely to buy cereals as well.

First data mining systems were specialized in one task and dedicated to one domain of interest. In recent years we observe a constant evolution of data mining systems to general purpose systems that are tightly coupled with database technology. This integration happens particularly in data warehouse environments which provide the best data source for analysis and discovery of patterns. Mining association rules in large data warehouses is a time-consuming task. Besides, the size of the set of discovered rules can easily exceed the size of the original database. One possible solution is to store discovered patterns in the database and query them as needed at some other point in time. Usually, users of decision support systems retrieve and penetrate iteratively the set of discovered patterns, investigating interesting rules in greater detail, scrutinizing customer transactions that either support or violate given rules, etc. Such queries require excessive subset searches and are not well-supported by traditional database management systems.

In this paper we present different indexing methods for supporting subset search in relational databases. We concentrate on queries that are typical for data mining and pattern post-processing in decision support systems. We discuss advantages and disadvantages of each index type and analyze its usefulness for general purpose data mining systems. Our discussion is illustrated by the results of experiments. We also provide a comparison of those indexes with traditional database access method, namely, the B+ tree index.

2 RELATED WORK

The problem of association rule mining was first introduced in (Agrawal et al., 1993). The paper identified the discovery of frequent itemsets as a key step in association rule mining. In (Agrawal and Srikant, 1994) the authors presented the basic algorithm, called *Apriori*, that quickly became the seed of several

other data mining algorithms. The idea of tight integration of knowledge discovery systems with the existing database framework was formulated in (Imielinski and Mannila, 1996).

Traditional databases provide several indexing techniques, e.g. B+ trees (Comer, 1979), bitmap indexes (Chan and Ioannidis, 1998) or R-trees (Guttman, 1984), but those techniques optimize exact match querying and single tuple access. Indexing of data items with set-valued attributes was seldom researched and resulted in development of signature files (Ishikawa et al., 1993), Russian Doll Trees (Hellerstein and Pfeffer, 1994), inverted files (Araujo et al., 1997) and hash-based indexes (Helmer, 1997). Similar work was done in the area of text retrieval (Baeza-Yates and Ribeiro-Neto, 1999). Another study of set-oriented indexes performance can be found in (Helmer and Moerkotte, 1999). First proposals for association rule retrieval from a relational database can be found in (Morzy and Zakrzewicz, 1998). The authors proposed a special structure, called group bitmap index, and its variation, called hash group bitmap index, that addressed the problem of the retrieval of set-valued attributes from a relational database.

This paper is organized as follows. Chapter 3 provides basic definitions of frequent itemsets, association rules and the description of data mining queries. Set-oriented indexes are described in Chap. 4. Chapter 5 contains the results of conducted experiments. We conclude with a brief summarization and future work agenda discussion in Chap. 6.

3 BASIC NOTIONS

3.1 Association Rules

Let $L = \{l_1, \ldots, l_n\}$ be a set of literals called *items*. Let D be a set of variable length transactions and $\forall T \in D : T \subseteq L$. We say that the transaction T supports an item x if $x \in T$. We say that the transaction T supports an itemset X if it supports every element $x \in X$. The *support* of an itemset is the number of transactions supporting the itemset. The problem of discovering frequent itemsets consists in finding all itemsets with the support higher than user-defined minimum support threshold denoted as *minsup*. An itemset with the support higher than minsup is called a *frequent itemset*.

An association rule is an implication of the form X $\rightarrow$ Y where X $\subset L$, Y $\subset L$ and X $\cap$ Y = $\emptyset$. X is called the *head* of the rule whilst Y is called the *body* of the rule. Two measures represent statistical significance and strength of the rule. The *support* of the rule is the number of transactions that support X $\cup$ Y. The *confidence* of the rule is the ratio of the number of transactions that support the rule to the number of transactions that support the head of the rule. We say that a customer transaction *satisfies* the rule if X $\cup$ Y are contained in the transaction. A customer transaction *violates* the rule if it contains X but does not contain Y.

The problem of discovering association rules consists in finding all rules with support and confidence higher than the user-specified thresholds of minimum support and confidence, called *minsup* and *minconf* respectively.

3.2 Storage Structures and Queries

Most customer transactions are stored in flat relations using two attributes: one describing the purchased item and another organizing items into sets. Additional attributes might represent the price of an item, the quantity of items purchased, the discount, etc. Such structure allows for efficient storage of varying length transactions. Association rules discovered in customer transactions can be stored in a relational database using two tables. First table contains rule identifiers and values of support and confidence for each rule. Second table contains rule identifiers and items contained in rule's head and body along with item type (head or body respectively). Figure 1 presents an example of database storage for association rules using tables *rules* and *items*.

rules

rule	support	confidence
1	0,45	0,87
2	0,72	0,45
3	0,64	0,69

items

rule	item	type
1	bread	head
1	butter	body
2	butter	head
2	cheese	body
3	bread	head
3	milk	head
3	cereals	body

Figure 1: Association rules stored in the database.

A query predicate P is defined by a set-valued attribute X, a set of itemsets Q and a set comparison operator $\Theta \in \{=, \subseteq, \supseteq\}$. Let q denote a finite set of items. Three classes of queries can be identified with respect to a set-valued attribute:

- equality query $\{T \in Q : T.X = q\}$
- subset query $\{T \in Q : T.X \supseteq q\}$
- superset query $\{T \in Q : T.X \subseteq q\}$

In this paper we focus on data mining applications. In association rules discovery two types of queries are common:

- select all customer transactions that contain a given set of items (e.g., to find transactions and customers who satisfy or violate a given rule)

- select all rules that contain a given set of items in their head or body (e.g., to analyze associations between given items in detail).

Both types of queries contain a subset search. This problem was studied in (Graefe and Cole, 1995) where the authors proposed a novel relational division operator. Currently available database systems do not implement either the relational division operator or the subset search operator. Hence, those queries are difficult to express in standard SQL language. To illustrate this problem below we present two queries that retrieve identifiers of itemsets that contain the set {*'milk'*, *'butter'*, *'cereals'*} from the table *Purchases*.

Query using several joins

```
SELECT DISTINCT A.set_id
FROM purchases A, purchases B,
purchases C
WHERE A.set_id = B.set_id
AND B.set_id = C.set_id
AND A.item = 'milk'
AND B.item = 'butter'
AND C.item = 'cereals';
```

Query using GROUP BY clause

```
SELECT set_id FROM purchases
WHERE item IN
  ('milk','butter','cereals')
GROUP BY set_id
HAVING count(*) = 3;
```

Both queries are time-consuming and costly in terms of query optimization. First query requires a join of three very large tables whilst second query requires sorting the relation and executing an aggregate function on all groups. Besides, the above queries are hard to read and they lack flexibility. Although traditional B+ trees indexes can be used to speed-up those queries, those indexes are row-oriented and in case of set containment queries they require multiple index scans. This example proves that traditional access methods are insufficient for excessive subset search queries in large databases and data warehouses. Therefore, several novel set indexing techniques were proposed. In the next chapter we present the description of several different indexes.

4 SET-ORIENTED INDEXING TECHNIQUES

4.1 Group Bitmap Index

Group bitmap index is used to speed up subset search and content-based retrieval of various sets. The main idea consists in creating a binary representation of each set and using resulting bitmaps to find relevant sets. A bitmap key is a binary number of length N, where N is the total number of distinct items in a database. For a given set the bitmap key has k*th* position bit set to '1' if the set contains item *k*. All bitmap keys are stored in an index table together with set identifiers they refer to. When a user issues a subset search query, a searched bitmap key is created for the searched set of items. Searched bitmap key is constructed in the same way as an indexed set bitmap key. Next, the searched bitmap key is compared to every entry in the index table by means of the bitwise AND operation. All index keys that contain '1's on the same positions as in the searched bitmap key are returned as the answer to the query. Figure 2 presents a sample customer transaction database and corresponding group bitmap index. When a user searches for customer transactions containing products *'milk'* and *'bread'* the searched set bitmap key is computed. This key has bits set to '1' on positions 1 and 12. This key is compared to all index table entries to find index keys that contain '1's on the same positions. As the result, the customer transaction 3 is returned.

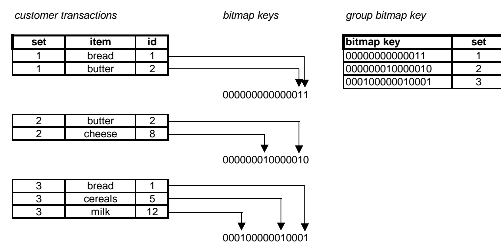

Figure 2: Group bitmap index.

The main drawback of the group bitmap index is its size and the fact, that the size of the bitmap key varies with the maximal number of distinct items in the database. In practice, the number of distinct items can be of order of tens or hundreds of thousands which makes group bitmap index long, sparse (99,99% of positions would be set to '0') and difficult to process. The number of distinct items is not constant either, so adding new items to the database could result in costly index reconstruction. On the other hand, for many databases, in particular when the number of distinct items is relatively small (say up to 1000 diffe-

rent items) and remains constant, the group bitmap index is the best indexing method available.

4.2 Restricted Group Bitmap Index

Since there are no data types that have the size of 1000 bits, the group bitmap index is usually implemented as an array of binary numbers of available type, e.g. 32 bit `long` (notice that the size of the type `long` may vary from one computer architecture to another). Indexed sets usually contain no more than 10–40 items (this is the average number of items purchased by customers). Hence, most entries in the array of bitmap keys for a given itemset are empty. Besides, most queries contain a small number of items, usually 2–5, which makes the bigger part of an index bitmap key unnecessary in the search. To avoid reading into memory and processing of unnecessary key parts we restricted a group bitmap index to only those parts of the bitmap key that are present in the searched bitmap key.

The advantage is the significant reduction of the index size. Unfortunately, this approach has a rather theoretical character because such index has to be reread into main memory for every query as queries differ in parts of the index key.

4.3 Hash Group Bitmap Index

Hash group bitmap index (Morzy and Zakrzewicz, 1998) tries to eliminate the shortcomings of the group bitmap index. Each hash bitmap key representing a customer transaction has a constant size of n that is much smaller than the number N of distinct items in the database. Hash bitmap key of a customer transaction is a bitwise sum of hash keys representing items contained in the transaction. Hash key of an item x is an n-bit binary number defined as

$$hash_key(x) = 2^{(x \ mod \ n)}$$

When a user issues a subset search query, the process is performed in two steps. First, in the *filtering step*, a searched hash bitmap key is computed. This hash bitmap key is then compared to all hash keys stored in the hash index table. This comparison is performed by the means of the bitwise AND operation. All sets that have '1' on the same positions as the searched hash bitmap key are returned as a result of the first step. These are candidate sets because due to the ambiguity of hashing some of those sets do not contain the searched subset. This happens because hash bitmap keys do not represent indexed sets uniquely. On the other hand, all itemsets that potentially contain the searched subset are present in the result after filtering. In the second step, called the *verification step*, itemsets returned by the filtering step are actually checked for the containment of the searched subset. This is usually done using traditional techniques because the number of itemsets to be verified remains relatively small.

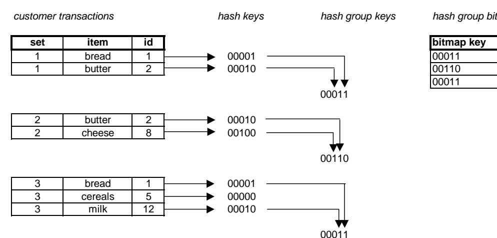

Figure 3: Hash group bitmap index.

Figure 3 presents a sample database of customer transactions and the creation of the hash group bitmap key of size $n = 5$. Figure 4 presents a subset search query evaluation using the hash group bitmap index. A user issues a query for all sets containing the set $\{$*'bread'*, *'butter'*$\}$. A searched hash bitmap key is created for this set. Next, all entries in the hash bitmap index are scanned. Bitwise AND operation is performed between the searched hash bitmap key and index entries. Entries representing sets 1 and 3 contain '1's on the same positions as in the searched hash bitmap key and therefore these entries are included in the set of candidate results. In the last step both sets are checked and the set 1 remains as the answer to the query.

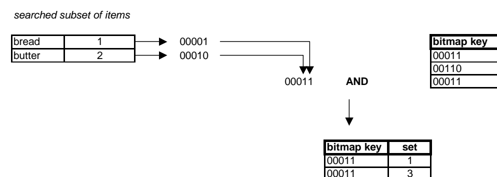

Figure 4: Subset search using hash group bitmap index.

The main advantage of the hash group bitmap index is the reduction of the size of a key and independence of the size of a key from the number of distinct items in the database. Unfortunately, the ambiguity of hashing is the source of false hits (itemsets considered to be candidate answers but pruned in the verification step). The need to check candidates in the database can seriously harm the performance of the index, particularly for the queries with low selectivity.

4.4 Signature Index

Signature index (Helmer, 1997; Nørvåg, 1999) is very similar to hash group bitmap index and uses the technique of superimposed bit vector coding. Each ele-

Table 1: Synthetic data parameters

parameter	symbol	value
number of itemsets	N_{trans}	10000 to 100000
number of different items	N_{items}	50 to 100
average number of items per set	T_{len}	5 to 10
size of the query	Q_s	1 to 8

ment in the indexed set receives an n-bit vector with k bits set to '1'. Next, all vectors representing items in the indexed set are superimposed using the bitwise OR operator to form set signatures. The number k of bits used to represent a single item strongly affects the performance of the index. Bigger values of k allow to represent more items (without the need to use hashing), but result in signatures which tend to be populated with many '1's, thus requiring the verification of many signatures in the database. Searching the signature index is similar to searching the hash group bitmap index. First, a search signature for the user's query is created by bitwise ORing vector representations of the items in the query. Next, the searched signature is compared to all signatures stored in the index. Signatures containing '1's on the same positions as in the searched signature are returned as candidates. In the verification step candidate sets are checked in the database for the containment of the searched items.

The main advantage of the signature index is the constant size of the signatures. This allows for easy maintenance and processing of index entries. Most characteristics of the signature index are similar to the characteristics exhibited by the hash group bitmap index. In our implementation we decided to make the signature much longer than the size of the hash key to see if this factor affects the overall performance of the index.

4.5 Russian-Doll Trees

The Russian-Doll Trees (RD-Trees) were first presented in (Hellerstein and Pfeffer, 1994). An RD-Tree is a modification of the well-known R-Tree (Guttman, 1984), an efficient index for spatial data. The structure of an RD-Tree is the following: the leaf nodes of the tree contain indexed sets (called *base sets*) and their descriptions (called *bounding sets*). A bounding set is the smallest set containing the base set and fulfilling certain conditions. Internal nodes of the RD-Tree contain pointers to the child nodes and their bounding sets. A bounding set in an internal node must contain all bounding sets of the child nodes. Searching the RD-Tree uses an inclusion property which states that if the searched set is not contained in the bounding set of a node, it cannot be contained in any child node of a node. Searching starts at the root of the RD-Tree and proceeds in a recursive manner to all nodes that bound the searched set. The bounding sets depend on the representation of keys in the leaf nodes. Several different techniques were proposed, among them complete representations, signatures, rangesets and combined representations. Compared to previous methods, RD-Tree are not dependent of the number of distinct items in a database. On the other hand, RD-Trees are much more expensive in terms of maintenance. Inserting a new set into the index requires finding nodes that need least enlargement to include a new set. Similarly, when a node overflows it must be split to make two new nodes such that two nodes are maximally disjoint.

4.6 Inverted Files

Inverted file, introduced in (Araujo et al., 1997), is an index structure consisting of two elements: the *vocabulary* and the *occurrences*. The vocabulary contains all items that appear in the indexed sets. For each item a list of sets in which this item appears is maintained. The set of all those lists is called the occurrences. Searching an inverted file consists in finding all searched items in the vocabulary, retrieving occurrence lists related to searched items and determining the intersection of all lists. Inverted files can also be used to solve context queries and approximate queries. Originally inverted files were used in text processing to index text collections. Inverted files exhibit similar characteristics to bitmap indexes which are available in many commercial database management systems.

5 Experimental Results

In our experiments we constrained ourselves to four types of set-oriented indexes, namely, to group bitmap indexes (both simple and restricted), hash group bitmap index and signature index. All experiments were conducted on Oracle 8.1.7 running under Linux (kernel 2.2.19) with two Pentium II 450 MHz processors and 512 MB memory. Data sets were created using DBGen generator from the Quest Project (Agrawal et al., 1994). Table 1 summarizes the values of different parameters that affect the characteri-

stics of the data sets used in our experiments. These data sets tend to mimic typical customer transactions in a supermarket. Number of distinct items is relatively small and represents the analysis at a higher level where associations between groups of products are discovered rather than associations between individual products. Here each item represents a product group such as beverages, bakery products, fruits, etc. The size of a single customer basket is also small because individual items were previously merged into product groups. Query sizes vary from general queries (considering one or two items) to very specific (considering several items). The number of customer transactions varied from 10000 (relatively small database) to 100000 (relatively big database).

Figure 5 presents the performance of the group bitmap index. This index scales almost linearly with the database size. It is slightly better suited to answer complex queries than simple queries. The execution times of queries using reduced group bitmap index are presented on Fig. 6. This index shares all characteristics of the group bitmap index but it is not affected by the length of the query. Query execution is faster because no empty index keys are processed. On the other hand, this index requires recreation for each query because for each query different parts of the index key table may become empty, so the overall performance of the reduced group bitmap index is affected by the time needed to recreate the index. Signature index characteristics are depicted on Fig. 7. This index exposes the best characteristics regarding both different database sizes and different query sizes. In our implementation we used long signatures and we did not introduce hashing. If the number of items exceeds the length of a signature, hashing becomes unavoidable. This hashing introduces ambiguity of hits and additional verifying phase is required to prune false hits. In that case a serious degradation of performance can be observed. Hash group bitmap index characteristics are presented on Fig. 8. In this implementation we used an opposite approach compared to signature index. We used very short hash keys consisting of one machine word (32 bits) imposing strong hashing. As it can be observed, under these circumstances the performance of the index is very bad in case of short queries. This is due to strong ambiguity of hashing, as a matter of fact more than 90% of hits were false hits and the majority of time was spent on filtering the result set and verifying hits. As the size of the query increases hash index performance rapidly grows and the number of false hits that need to be verified drops significantly.

Figures 9 and 10 present the comparison of query execution times for different indexes and different database sizes for queries of lengths 1 and 4 respectively. We measured also the performance of the traditional B+ tree index but we omitted it in the final results because for all database sizes and for all query lengths traditional indexing technique was significantly inferior to bitmap and signature indexes (on average all queries were 10 times slower using B+ tree indexes). For query lengths 1 and 4 we also omitted hash group bitmap index because of the unacceptable response times (but please be aware of the fact that in our experiment we used only the shortest possible hash key). It can be observed that the best results can be achieved using signature and reduced bitmap indexes. Figure 11 shows the comparison of execution times for queries of length 8. Hash group bitmap index is still the slowest one, but the difference is not that big. Besides, in many real world applications indexes with hard-coded maximal number of items (such as bitmap and reduced bitmap indexes) are unsuitable. It is worth noticing that the performance of flexible hash group bitmap index is not significantly worse from the previous indexes. The sizes of different types of indexes are presented on the Fig. 12. All indexes grow linearly with the database size with hash group bitmap index being the smallest. This feature also proves the practical usability of hash-based index because due to its small size it can easily fit into the available main memory, making index search and processing much faster.

6 CONCLUSION

In this paper we investigated the performance of four different indexing techniques that facilitate content-based retrieval and subset search in relational databases. Our experiments focused on data sets and queries that are characteristic for data mining applications. We showed that traditional access methods, i.e. B+ trees, are unsuitable for this class of queries and that additional indexing techniques must be developed. Conducted experiments proved that for many real world problems bitmap group indexes and signature indexes provide efficient solution to the subset search problem. We also noticed that for applications that require flexible indexing techniques hash group bitmap indexes may be successfully used.

Still, many questions remain unanswered. First of all, our experiments were conducted on top of the database management system and not within it. Incorporating novel index types into the core of the database system could result in larger performance gain for all types of indexes. We concentrated on subset queries, but other types of queries should be also efficiently supported, e.g. superset queries or overlap queries. Another question is what type of statistics and histograms are required for cost-based optimizer module to make efficient use of new set-oriented indexes. New database systems allow the users to define

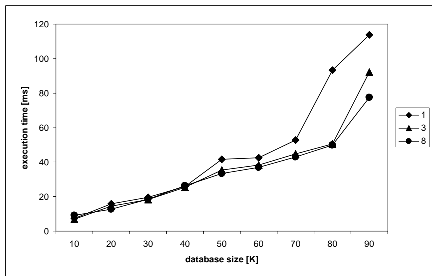

Figure 5: Group Bitmap Index

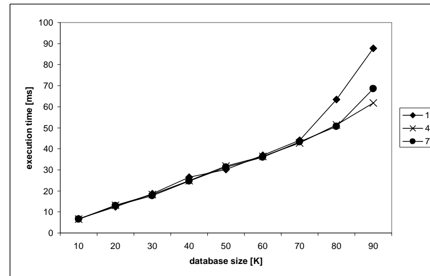

Figure 6: Reduced Group Bitmap Index

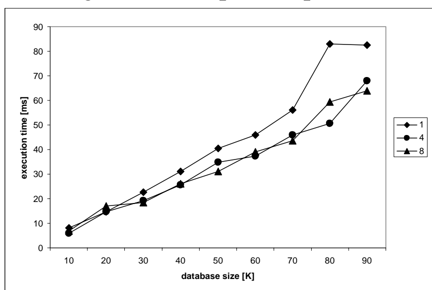

Figure 7: Signature Index

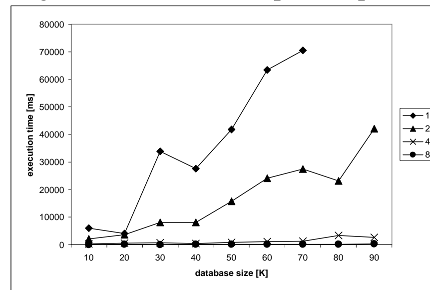

Figure 8: Hash Group Bitmap Index

their own abstract data types, among them set and collection types (in the form of nested tables or varrays). The issue of indexing of ADTs remains a challenging research area.

REFERENCES

Agrawal, R., Carey, M. J., Faloutsos, C., Ghosh, S. P., Houtsma, M. A. W., Imielinski, T., Iyer, B. R., Mahboob, A., Miranda, H., Srikant, R., and Swami, A. N. (1994). Quest: A project on database mining. In Snodgrass, R. T. and Winslett, M., editors, *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, page 514, Minneapolis, Minnesota. ACM Press.

Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining association rules between sets of items in large databases. In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB)*, pages 487–499. Morgan Kaufmann.

Araujo, M. D., Navarro, G., and Ziviani, N. (1997). Large text searching allowing errors. In Baeza-Yates, R., editor, *Proceedings of the 4th South American Workshop on String Processing*, pages 2–20, Valparaiso, Chile. Carleton University Press.

Baeza-Yates, R. and Ribeiro-Neto, B. (1999). *Modern information retrieval*. Addison-Wesley.

Chan, C. Y. and Ioannidis, Y. E. (1998). Bitmap index design and evaluation. In Haas, L. M. and Tiwary, A., editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 355–366, Seattle, Washington. ACM Press.

Comer, D. (1979). The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137.

Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R. (1996). *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press.

Graefe, G. and Cole, R. L. (1995). Fast algorithms for universal quantification in large databases. *TODS*, 20(2):187–236.

Guttman, A. (1984). R-trees: A dynamic index structure for spatial searching. In Yormark, B., editor, *SIGMOD'84, Proceedings of Annual Meeting*, pages 47–57, Boston, Massachusetts. ACM Press.

Hellerstein, J. M. and Pfeffer, A. (1994). The rd-tree: An index structure for sets. Technical Report 1252, University of Wisconsin at Madison.

Helmer, S. (1997). Index structures for databases containing data items with setvalued attributes. Technical Report 2/97, Universität Mannheim.

Helmer, S. and Moerkotte, G. (1999). A study of four index structures for set-valued attributes of low cardinality. Technical Report 2/99, Universität Mannheim.

Imielinski, T. and Mannila, H. (1996). A database perspective on knowledge discovery. *CACM*, 39(11):58–64.

Ishikawa, Y., Kitagawa, H., and Ohbo, N. (1993). Evaluation of signature files as set access facilities in oodbs.

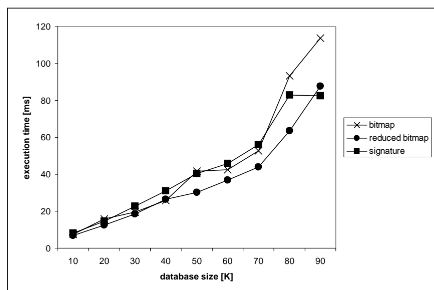

Figure 9: Short queries

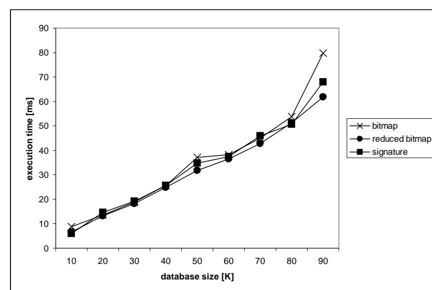

Figure 10: Medium queries

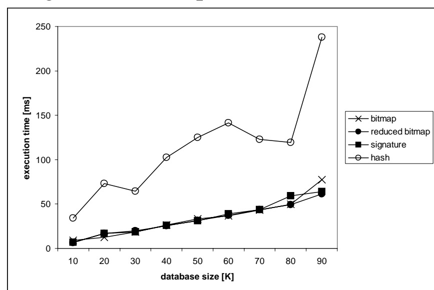

Figure 11: Long queries

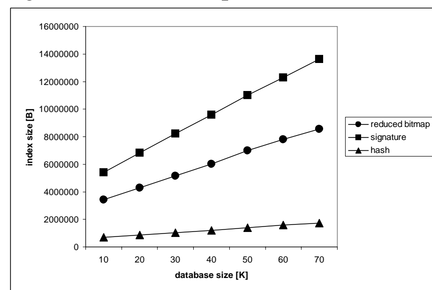

Figure 12: Index size

In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 247–256, Washington, D.C. ACM Press.

Morzy, T. and Zakrzewicz, M. (1998). Group bitmap index: A structure for association rules retrieval. In Agrawal, R., Stolorz, P. E., and Piatetsky-Shapiro, G., editors, *Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 284–288, New York, USA. ACM Press.

Nørvåg, K. (1999). Efficient use of signatures in object-oriented database systems. In Eder, J., Rozman, I., and Welzer, T., editors, *Proceedings of the 3rd East European Conference on Advances in Databases and Information Systems (ADBIS)*, volume 1691 of *Lecture Notes in Computer Science*, pages 367–381, Maribor, Slovenia. Springer.