A Hybrid Index Structure for Set-Valued Attributes Using Itemset Tree and Inverted List*

Shahriyar Hossain and Hasan Jamil

Department of Computer Science
Wayne State University, Michigan, USA
shah_h@wayne.edu, jamil@cs.wayne.edu

Abstract. The use of set-valued objects is becoming increasingly commonplace in modern application domains, multimedia, genetics, the stock market, etc. Recent research on set indexing has focused mainly on containment joins and data mining without considering basic set operations on set-valued attributes. In this paper, we propose a novel indexing scheme for processing *superset*, *subset* and *equality* queries on set-valued attributes. The proposed index structure is a hybrid of itemset-transaction set tree of "frequent items" and an inverted list of "infrequent items" that take advantage of the developments in itemset research in data mining. In this hybrid scheme, the expectation is that basic set operations with frequent low cardinality sets will yield superior retrieval performance and avoid the high costs of construction and maintenance of item-set tree for infrequent large item-sets. We demonstrate, through extensive experiments, that the proposed method performs as expected, and yields superior overall performance compared to the state of the art indexing scheme for set-valued attributes, i.e., inverted lists.

1 Introduction

The need for containment queries involving set-valued attributes is found in a variety of application areas, ranging from scientific databases to XML documents, annotation databases, market basket analysis, production models, multimedia [4,6], bio-molecular databases [1], etc. Containment queries span a wide range of query families, from simple existence queries to composite similarity, pattern matching, or graph isomorphism queries. For example, against a movie annotation database we could ask *"find all the movies where Tom Hanks and Meryl Streep both performed"*, or *"find all users who visited sports or finance pages but nothing else"* in a database of internet usage.

To evaluate containment queries, namely, *subset*, *superset* and *equality* efficiently, we need targeted access to data using index structures. The state of the art method for indexing set-valued databases is inverted list [12]. The problem with inverted lists, however, is that for frequent items, they grow quite large. As

* This research was partially supported by National Science Foundation grants CNS 0521454 and IIS 0612203.

P. García Bringas et al. (Eds.): DEXA 2010, Part I, LNCS 6261, pp. 349–357, 2010.
© Springer-Verlag Berlin Heidelberg 2010

frequent items are queried frequently and regularly, the performance for query evaluation will degrade for very large databases with items of skewed distribution. In order to avoid the intersection of large inverted lists, we can actually devise an index structure where some such intersections are pre-computed and stored in memory or secondary storage. The idea of pre-computation is borrowed from existing research in spatial databases [11], and data mining [10].

Our contribution is summarized in the following way:

- We propose an index structure for set-valued attributes, called Mixed Inverted List Itemset-Tidset index or *MixIIT*. This is a hybrid of concept lattice and inverted lists which is kept in secondary storage. However, in MixIIT, we augment the concept lattice with itemset-tidset lists.
- We develop novel and efficient algorithms for computing basic set operations such as subset, superset and equality that leverages the proposed *MixIIT* structure. The details of the algorithms may be found in [13].
- We experimentally compare the performance of *MixIIT* with inverted list and show that not only does MixIIT perform superior to inverted lists, the low memory requirement of *MixIIT* makes it possible to exploit this structure for practical applications today.

The rest of the paper is organized as follows. We briefly discuss recent and relevant research in section 1.1. In Section 2, we present the *MixIIT* index structure and in section 3, we describe the query evaluation algorithms. An extensive experimental evaluation is discussed in section 4 that compares MixIIT with the inverted lists index. Finally, section 5 summarizes our research.

1.1 Related Research

Reported research on the evaluation of basic containment queries with set-valued predicates are few and far between. So far, database research has mostly focused on similarity [17] and join queries [16]. The research on set containment queries can be divided into three major categories. First, we have signature based approaches where the transactions of varying lengths are converted into signature bit-strings of fixed length [7]. The greatest advantage of signature based methods is that set comparison operations such as subset, superset checking get reduced to simple bitwise operation. Signatures can be organized into various indexing structures such as Sequential Signature Files [15], Bit-Slice Signature file (BSSF), Multilevel Signature file, Compressed Multi Framed Signature file, S-Tree and its variants, Signature Graph and Signature tree [5] The common problem with any signature based index is that signatures can only generate the candidate set of transactions which covers the query result. As a result, there will always be some false positives.

The second approach in indexing set-valued databases originated from information retrieval research. Inverted files or list consist of a directory containing all distinct items that can be searched for, and a list for each distinct item which contains the transactions where the item appeared [3]. As shown in [12], inverted

list based indexing techniques perform better than signature file based ones for different database, set, vocabulary and query cardinality. But, transaction lists for frequent items may grow quite large which may eventually slow down the query evaluation process.

In [11], Hellerstein et. al. proposed a R-tree [9] based indexing method for set-valued databases. Computing bounding box for itemsets is not as simple as that for spatial data points. Also, storing bounding box of unequal lengths creates maintainability issues. In [19], Terrovitis et al proposed an indexing technique which combines the advantages of inverted lists with the power of in-memory index for frequent itemset. However, it is not possible to update the FP-tree [10] based *access tree* online.

2 Index Structure

In this section, we introduce a hybrid index that combines a main memory itemset-tidset representation of concept lattice with an inverted list residing in secondary storage. First, we give background information for inverted lists and itemset-tidsets and explain their benefits and drawbacks. Then, we show how these indexing schemes are combined in the set indexing system.

Table 1. Example Database

(a) Vocabulary

Items	A	C	D	T	W

(b) Transaction Database

Transaction	1	2	3	4	5	6
Items	ACTW	CDW	ACTW	ACDW	ACDTW	CDT

2.1 Inverted File Index

The inverted file index is essentially a two dimensional list. The first dimension is a list of all the distinct items appearing in the database. Each node in the vocabulary list points to a list of transactions where the item appeared. For containment query evaluation, one needs to store the length of every transaction along with the transaction id.

The evaluation of set based queries on inverted file index may be trivial but maintaining an inverted list is complicated. The lists may be huge for large databases as the id of a transaction is inserted as many times as the number of items it contains. As a result, theoretically, the size of the inverted file could be similar to the size of the transaction collection or even larger. Due to their size, the inverted lists are stored in secondary storage. Therefore, the larger these inverted lists are, the more pages have to be retrieved from the disk for evaluating a query. Moreover, the most frequent items will have the longest inverted lists. This is particularly damaging for the evaluation of set-valued queried, as the topmost frequent items are usually the ones most frequently queried.

2.2 Itemset-tidset Index

Concept lattices are used in many application areas to represent conceptual hierarchies among objects in the underlying data. The field of Formal Concept Analysis [8] has grown to a powerful theory for data analysis, information retrieval and knowledge discovery. There is an increasing interest in the application of concept lattices for data mining, especially for mining association rules [14] and generating frequent itemset [20,23]. Despite their numerous applications, concept lattices have never been employed for indexing set valued databases. In this section, we will present the conceptual model of lattice based index structure. We begin the discussion by defining some of the topics for formal concept analysis from [21] in the light of set valued databases.

Definition 1. A formal concept is a pair (A, B) with $A \subseteq G$, $B \subseteq M$, $A' = B$ and $B' = A$. (This is equivalent to $A \subseteq G$ and $B \subseteq M$ being maximal with $A \times B \subseteq I$.) A is called extent and B is called intent of the concept.

It is easy to see that even for a small vocabulary the number of concepts can grow very large. One possible way of combating this growth is to store only the most frequently occurring elements as in closed itemsets [22]. In the context of concept lattices, and for real-life databases, an iceberg [18] type solution seems appropriate where only the frequent items are stored in the lattice and less frequent items are still kept in an inverted list like structure so that the membership of the items in these two structures can be adjusted (following database updates) when the items become less or more frequent. The index structure discussed next based on concept lattice and inverted list captures this spirit.

2.3 Mixed Inverted List Itemset-Tidset (*MixIIT*) Index

In *MixIIT* index structure we store the frequent items in the database in the Itemset-Tidset search tree built essentially using the algorithm in [22]. However, we deviate from the algorithm in the following principal ways:

– To facilitate subset queries, all the edges are considered bidirectional.
– We avoid strict support based pruning of the itemset-tidset as proposed in [22]. For example, let item A and B passes the support threshold but, itemset AB falls below the cutoff. If we remove itemset AB as proposed in [22], then we will not be able to evaluate queries containing AB correctly. So, in the lattice structure, we store *all* the concepts which can be constructed with the frequent items, not just the ones which pass the support threshold.
– Finally, we define a special concept, called *sink*, which contains all the frequent items in its intent. The sink concept is connected with all the leaf nodes of the concept tree. This arrangement is necessary for the evaluation of the subset queries where we need to start from the most general intent and subsequently look for the specific one.

As the extent of every concept can grow quite large, we store them in a secondary storage, which can serve as the file system or an SQL database. Itemset-tidset tree of only frequent items will generate incorrect answers for subset and superset queries with infrequent items in the query set. We get around this issue by maintaining the inverted list of transactions for the infrequent items in the secondary storage. In Figure 1(a), we show the *MixIIT* index for the example database in Table 1. We create an itemset-tidset tree for the top three frequent items. The three items, namely A, D and T equally deserve the position for third frequent item. We made a random choice A. Also, ACW is the closure of A and AC. So, using the closure properties described in [22], we can replace A and AC from the tree with their closure. Finally, we added links from the sink concept ACW to all the leaves of the tree which, in this example, is only CW. The inverted lists of infrequent items D and T are stored in the secondary storage.

The intents are stored in bit string format as shown in Figure 1(b). This representation is different from signature as the transformation between itemset and its corresponding bit string is one to one. With bitstring representation, we can compute subset or superset with a bitwise comparison instead of set manipulation. For example, if a and b are the bit string representation of itemset A and B, respectively and if, $A \subseteq B$ then $a \wedge b = a$.

3 Query Processing Using MixIIT

In this section, we present the generic approach taken for evaluating the three types of queries we are interested in: subset, equality and superset. In [13], we present a more detailed discussion on the evaluation process.

The evaluation algorithms for all three types of queries have two main stages: (a) evaluation in the itemset-tidset tree, and (b) evaluation in the inverted file. The frequent items from the query set are used to traverse the concepts in the itemset-tidset tree. A set of candidate transaction ids is created in the process. Next, the infrequent items are used to extract transactions ids from the inverted list. In the final stage, two lists of transaction ids are combined based on the nature of the query. The basic idea is that we use the intent of the concepts in the itemset-tidset tree to quickly trace a candidate answer to the query. The benefit is quite significant since we can avoid costly union and intersection operations between large transaction sets.

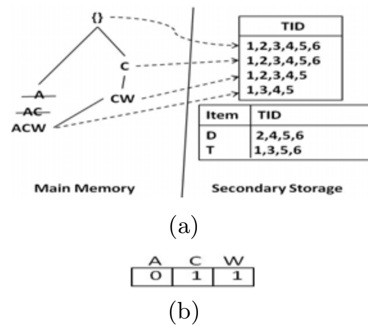


Fig. 1. (a) *MixIIT* index for the database in Table 1. (b) Bit String representation of the itemset CW.

4 Experimental Evaluation

We decided to compare the performance of *MixIIT* with inverted list based indices because in [12], the authors have shown that inverted lists perform better than signature based index structures for low cardinality set valued attributes, and even outperforms B-trees for containment queries in RDBMS [24].

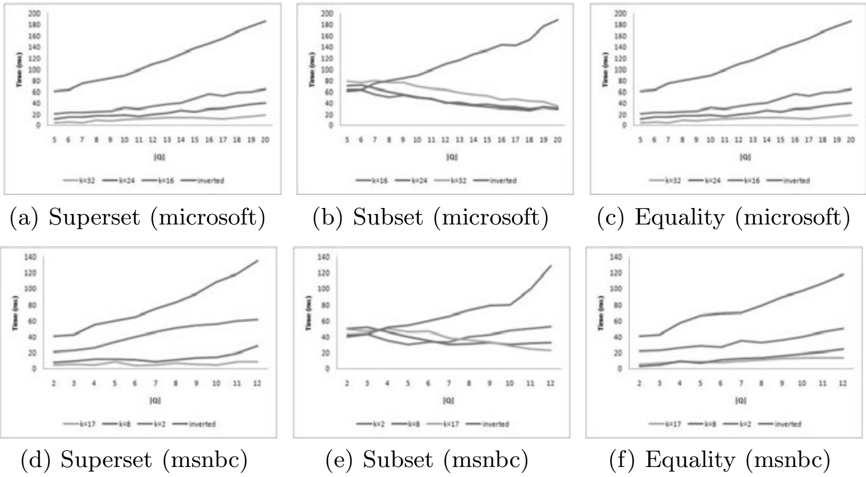
4.1 Data Sets and Performance Evaluation

For evaluation purposes, we use two real data sets from UCI KDD archive [2]. The first data set is a one-week log tracing the virtual areas that users visited in the web portal `www.microsoft.com`. Each record corresponds to a user session and the set value comprises the areas visited. There are 32k records and the vocabulary of the data set contains 294 distinct items (areas). The distribution of the items in the records is skewed and the average size of the record is 3 items. on the web portal of `msnbc.com` taken from the UCI KDD archive as well. The vocabulary here is very limited, comprising only 17 distinct items and unlike the previous one, the distribution of the items is relatively uniform. The average size of the record is 5.7 items. The total number of transactions for this data set is 989818. As a result, we end up with long inverted lists.

Query Evaluation with `microsoft` data. We created 15 queries with lengths varying from 5 to 20. The items from query were selected randomly from the vocabulary. In figure 2(a) through 2(c), we present the performance graph for subset, superset and equality query evaluation for `microsoft` data set. The data points in the plot depict average time required for queries of different lengths. For equality and superset queries, *MixIIT* outperforms inverted lists by a significant factor. The performance gain increases with the increase of $|Q|$. *MixIIT* structure performs better with the increase of frequent items in the Itemset-Tidset tree, as expected. We have mixed results for subset queries (figure 2(b)). For small query sets, *MixIIT* is not as efficient as inverted list and with the increase of frequent items, k, the performance degrades. As we discuss in [13], smaller query sets result in more choices for the recursive call. More frequent items in the lattice increases the number of leaf nodes and consequently, the number of upward links from the sink node increases. Hence, the performance degrades in both the cases. an increase in the number of frequent items stored, k from 16 to 32 there is an increase in the number of concepts in the itemset-tidset tree. There is also an increase in the number of leaves in the itemset-tidset tree from 1698 to 5678. So, there are more choices going upwards from the sink concept $ACDTW$ and more paths to traverse for any given subset query. As a result, with the increase in k, the performance will most likely degrade.

Query Evaluation with `msnbc` data. We created 10 random queries with lengths varying from 2 to 12. The threshold value for the memory resident lattice was varied from 2 to 8 and 17, which covers all the items in the vocabulary. The main contribution of this experiment is that we could evaluate the *MixIIT*

structure against a small yet real life data set which can be indexed solely with in-memory lattice. The findings are presented in figure 2(d) through 2(f). One of the notable difference from the `microsoft` data is the behavior of $k = 2$ threshold. At such a low threshold, the combination of itemset-tidset tree and inverted list is dominated by the performance of the inverted list. Another interesting observation is that the performance graphs of the inverted list as well as top 2 and 8 item *MixIIT* structures show an upward trend for large queries. This data set contains longer inverted lists for infrequent items compared to `microsoft` data set as there are more transactions for smaller vocabulary. As a result, the performance of the index structures, which depends on the inverted list begins to deteriorate with increasing query size.



(a) Superset (microsoft) (b) Subset (microsoft) (c) Equality (microsoft)

(d) Superset (msnbc) (e) Subset (msnbc) (f) Equality (msnbc)

Fig. 2. Performance comparison of *MixIIT* index with inverted list. (a-c): for *microsoft* data. (d-e): for *msnbc* data. Inverted list performance is shown as the top most curve in each graph.

5 Summary

In this paper, we have proposed a novel indexing scheme, *MixIIT*, which combines a main memory resident itemset-tidset tree with an inverted file for infrequent items, kept in secondary storage. We also introduced novel evaluation algorithms for subset, superset and equality queries using the *MixIIT* index. Finally, we performed an experimental study comparing *MixIIT* with inverted lists for these queries. We found that in the case of superset and equality queries, *MixIIT* clearly outperforms the inverted lists with reasonable main-memory overhead. Although much lower than inverted lists, the costs for subset queries in *MixIIT* appears to be a bit higher for smaller k, and showed trends of improvement as k grew larger. The primary goal of subset query is to find the minimal coverage of the frequent items in the query set. As a future work, we

wish to use a hash or B^+-tree based secondary structure which will help us find the minimal coverage without traversing the tree. We will also investigate how other forms of queries such as set intersection, union, join etc. can be computed with *MixIIT* structure. Finally based on the experimental results, it is apparent that relaxing the threshold for items in the lattice will not always guarantee better performance. It may even get worse, as we discovered for the subset queries. We plan to establish the relationship among the size of the possible set elements, number of records in the database, and the size of the sets allowed in a record so that an optimum threshold can be selected for best performance.

References

1. Bairoch, A., Apweiler, R.: The swiss-prot protein sequence data bank and its supplement trembl. Nucleic Acids Res. 27, 49–54 (1997)
2. Bay, S.D., Kibler, D., Pazzani, M.J., Smyth, P.: The uci kdd archive of large data sets for data mining research and experimentation. SIGKDD Explor. Newsl. 2(2), 81–85 (2000)
3. Bertino, E., Tan, C.K.-L., Ooi, B.C., Sacks-Davis, R., Zobel, J., Shidlovsky, B.: Indexing Techniques for Advanced Database Systems, pp. 151–184. Kluwer, Dordrecht (1997)
4. Böhm, K., Rakow, T.C.: Metadata for multimedia documents. SIGMOD Record 23, 21–26 (1994)
5. Chen, Y.: On the signature tree construction and analysis. TKDE 18(9), 1207–1224 (2006)
6. Jain, R., Hampapur, A.: Metadata in Video Databases. SIGMOD Record 23(4), 27–33 (1994)
7. Faloutsos, C.: Signature files. In: Information Retrieval: Data Structures & Algorithms, pp. 44–65 (1992)
8. Ganter, B., Stumme, G., Wille, R.: Formal concept analysis: Theory and applications 10(8), 926–926 (2004)
9. Guttman, A.: R-trees: a dynamic index structure for spatial searching. Morgan Kaufmann Publishers Inc., San Francisco (1988)
10. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. DMKD 8(1), 53–87 (2004)
11. Hellerstein, J.M., Pfeffer, A.: The RD-tree: an index structure for sets. Technical Report 1252, University of Wisconsin at Madison (1994)
12. Helmer, S., Moerkotte, G.: A performance study of four index structures for set-valued attributes of low cardinality. The VLDB Journal 12(3), 244–261 (2003)
13. Hossain, S., Jamil, H.: MixIIT: A hybrid index structure for set-valued attributes. Technical report, Wayne State University, USA (2010)
14. Hu, K., Lu, Y., Shi, C.: Incremental discovering association rules: A concept lattice approach. In: Zhong, N., Zhou, L. (eds.) PAKDD 1999. LNCS (LNAI), vol. 1574, pp. 109–113. Springer, Heidelberg (1999)
15. Ishikawa, Y., Kitagawa, H., Ohbo, N.: Evaluation of signature files as set access facilities in oodbs. In: SIGMOD, pp. 247–256. ACM Press, New York (1993)
16. Mamoulis, N.: Efficient processing of joins on set-valued attributes. In: SIGMOD, pp. 157–168 (2003)
17. Mamoulis, N., Cheung, D., Lian, W.: Similarity search in sets and categorical data using the signature tree. In: ICDE, March 2003, pp. 75–86 (2003)

18. Stumme, G., Taouil, R., Bastide, Y., Lakhal, L.: Conceptual clustering with iceberg concept lattices. In: Proc. of GI-Fachgruppentreffen Maschinelles Lernen'01, Universität Dortmund (2001)
19. Terrovitis, M., Passas, S., Vassiliadis, P., Sellis, T.: A combination of trie-trees and inverted files for the indexing of set-valued attributes. In: CIKM, pp. 728–737 (2006)
20. Valtchev, P., Missaoui, R., Godin, R.: A framework for incremental generation of closed itemsets. Discrete Appl. Math. 156(6), 924–949 (2008)
21. Wille, R.: Restructuring lattice theory: An approach based on hierarchies of concepts. In: Rival, I. (ed.) Ordered Sets, September 1981. NATO Advanced Study Institute, vol. 83, pp. 445–470 (1981)
22. Zaki, M.J., Hsiao, C.-J.: Charm: An efficient algorithm for closed itemset mining. In: SDM (2002)
23. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: KDD, pp. 283–286. AAAI Press, Menlo Park (1997)
24. Zhang, C., Naughton, J.F., DeWitt, D.J., Luo, Q., Lohman, G.M.: On supporting containment queries in relational database management systems. In: SIGMOD, pp. 425–436 (2001)