IMine: Index Support for Item Set Mining in Item Set Extraction

T.Senthil Prakash
PhD Research Scholar, PRIST University, Thanjavur

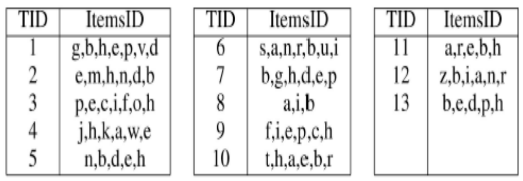
Dr.P.Thangaraj
Professor Head, Dept of CSE, Bannari Amman Institute of Technology, Sathyamangalam

ABSTRACT
Relational database management systems manages the tables with predefined indexes. In RDBMS indexes are created on the basis of attribute values. The Imine indexing scheme is used to index item sets in relational databases. The index creation is performed with out any constraints. IMine provides a complete representation of the original database. The Imine indexing scheme reduces the input/output cost for item set extraction and management process. The Imine index method supports different item set extraction algorithms. Different rule mining algorithms are supported by Imine index scheme. At present the Imine index scheme is developed under PostgreSQL DBMS.
The item set extraction and indexing operations are integrated in the system. The IMine scheme is improved to handle incremental data. In the incremental data handling mechanism the item sets and indexes are updated with respect transactional database changes. The data add, modify and remove operations are supported by the proposed index method. Reindexing process is optimized. Data structure is updated to handle all data distribution. The proposed item set extraction and indexing scheme is designed for the Oracle relational database. The system development is planned using J2EE environment.
General and compact structure - Provide tight integration of item set extraction - Can be efficiently exploited by different item set extraction algorithm - In particular, FP-growth and LCM v.2 - Has been integrated into the PostgreSQL DBMS

KEYWORDS
IMine, CFP Tree, I-B Tree, LCM, FP-Growth

1 INTRODUCTION
Relational DBMSs exploit indices, which are ad hoc data structures, to enhance query performance and support the execution of complex queries. A similar approach is proposed to support data mining queries. The Imine index is a novel data structure that provides a compact and complete representation of transactional data supporting efficient item set extraction from a relational DBMS. It is characterized by the following properties:
1. It is a covering index. No constraint is enforced during the index creation phase. Hence, the extraction can be performed by means of the index alone, without accessing the original database. The data representation is complete and allows reusing the index for mining item sets with any support threshold.
2. The IMine index is a general structure which can be efficiently exploited by various item set extraction algorithms. These algorithms can be characterized by different in-memory data representations and techniques for visiting the search space. Data access functions have been devised for efficiently loading in memory the index data. Once in memory, data is available for item set extraction by means of the algorithm of choice. The IMine index is evaluated with FP-growth and LCM v.2. Furthermore, the IMine index also supports the enforcement of various constraint categories.
3. The IMine physical organization supports efficient data access during item set extraction. Correlation analysis allows to discover data accessed together during pattern extraction. To minimize the number of physical data blocks read during the mining process, correlated information is stored in the same block.
4. IMine supports item set extraction in large data sets. A direct writing technique is exploited to avoid representing in memory the entire large data set. Direct materialization has a limited impact on the final index size because it is applied only on a reduced portion of the data set.

2 IMINE INDEX SCHEME
The transactional data set D is represented, in the relational model, as a relationR. Each tuple inRis a pair (TransactionID, ItemID). The IMine index provides a compact and complete representation of R. Hence, it allows the efficient extraction of item sets from R, possibly enforcing support or other constraints. We present the general structure of the IMine index. The physical organization of the index is presented together with a discussion of access cost.



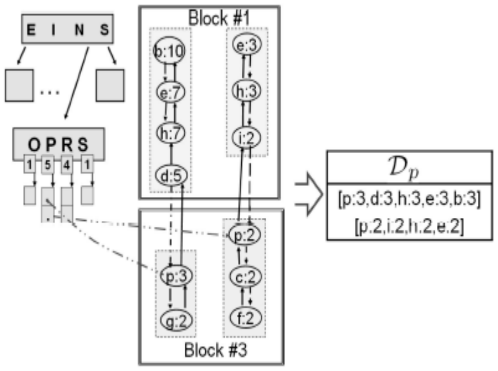
TID	ItemsID	TID	ItemsID	TID	ItemsID
1	g,b,h,e,p,v,d	6	s,a,n,r,b,u,i	11	a,r,e,b,h
2	e,m,h,n,d,b	7	b,g,h,d,e,p	12	z,b,i,a,n,r
3	p,e,c,i,f,o,h	8	a,i,b	13	b,e,d,p,h
4	j,h,k,a,w,e	9	f,i,e,p,c,h		
5	n,b,d,e,h	10	t,h,a,e,b,r		

2.1 IMine Index Structure
The structure of the IMine index is characterized by two components: the Item set-Tree and the Item-Btree. The two components provide two levels of indexing. The Item set-Tree (I-Tree) is a prefix-tree which represents relation R by means of a succinct and lossless compact structure. The Item-Btree (I-Btree) is a B+Tree structure which allows reading selected I-Tree portions during the extraction task. For each item, it stores the physical locations of all item occurrences in the I-Tree. Thus, it supports efficiently loading from the I-Tree the transactions in R including the item. In the following, we describe in more detail the I-Tree and the I-Btree structures.

2.1.1 I-Tree
An effective way to compactly store transactional records is to use a prefix-tree. Trees and prefix-trees have been frequently used in data mining and data warehousing indices, including

cube forest, FP-tree, H-tree, Inverted Matrix, and Patricia-Tries. Our current implementation of the I-Tree is based on the FP-tree data structure, which is very effective in providing a compact and lossless representation of relation R. However, since the two index components are designed to be independent, alternative I-Tree data structures can be easily integrated in the IMine index.

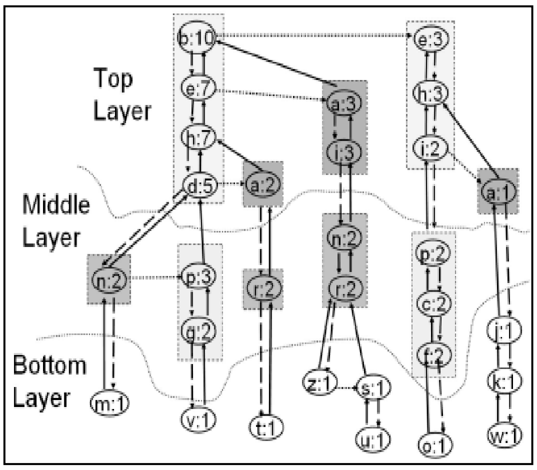
The I-Tree associated to relation R is actually a forest of prefix-trees, where each tree represents a group of transactions all sharing one or more items. Each node in the I-Tree corresponds to an item in R. Each path in the I-Tree is an ordered sequence of nodes and represents one or more transactions in R. Each item in relation R is associated to one or more I-Tree nodes and each transaction in R is represented by a unique I-Tree path.



It reports a small data set used as a running example, and the complete structure of the corresponding IMine index. In the I-Tree paths, nodes are sorted by decreasing support of the corresponding items. In the case of items with the same support, nodes are sorted by item lexicographical order. In the I-Tree, the common prefix of two transactions is represented by a single path. For instance, consider transactions 3, 4, and 9 in the example data set. These transactions, once sorted as described above, share the common prefix [e:3,h:3], which is a single path in the I-Tree. Node [h:3] is the root of two sub paths, representing the remaining items in the considered transactions.

Each I-Tree node is associated with a node support value, representing the number of transactions which contain (without any different interleaved item) all the items in the sub path reaching the node. For example, in sub path [e:3, h:3], the support of node [h:3] is 3. Hence, this sub path represents three transactions. Each item is associated to one or more nodes. The item support is obtained by adding the support of all nodes including the item.

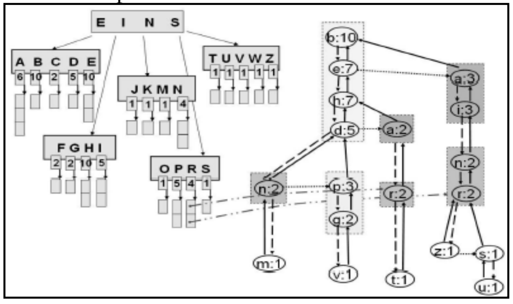
Nodes in the I-Tree are linked by means of pointers which allow selectively loading from disk the index portion necessary for the extraction task. Each node contains three pointers to nodes in the tree. Each pointer stores the physical location of the corresponding node. An arbitrary node includes the following links: 1) Parent pointer. 2) First child pointer (dashed edge linking node [p:3] to node [g:2]). When a node has more direct descendants, this pointer points to the first child node inserted in the I-Tree. 3) Right brother pointer. When a node has many brothers, the pointer points to the first brother node inserted in the I-Tree after the current node. These pointers allow both bottom-up and top-down tree traversal, thus enabling item set extraction with various types of constraints.



The I-Tree is stored in the relational table TI_Tree, which contains one record for each I-Tree node. Each record contains node identifier, item identifier, node support, and pointers to the parent, first child, and right brother nodes. Each pointer stores the physical location of the record in table TI_Tree representing the corresponding node.

2.1.2 I-Btree

The I-Btree allows selectively accessing the I-Tree disk blocks during the extraction process. It is based on a B+Tree structure. The I-Btree for the example data set and a portion of the pointed I-Tree. For each item i in relation R, there is one entry in the I-Btree. In particular, the I-Btree leaf associated to i contains i's item support and pointers to all nodes in the I-Tree associated to item i. Each pointer stores the physical location of the record in table TI_Tree storing the node. The pointers to the I-Tree nodes associated to item r.



2.2 IMine Data Access Methods

The IMine index structure is independent of the adopted item set extraction algorithm. Hence, different state-of-theart algorithms may be employed, once data has been loaded in memory. The in-memory representation suitable for the selected extraction algorithm is employed. Depending on the enforced support and/or item constraints and on the selected algorithm for item set extraction, a different portion of the IMine index should be accessed. Wedevised three data access methods to load from the IMine index the following projections of the original database: 1) Frequent-item based projection, to support projection-based algorithms. 2) Support-based projection, to support level based, and array-based algorithms. 3) Item-based projection, to load all transactions where a specific item occurs, enabling constraint enforcement during the extraction process.

Since IMine is a covering index, the original database is never accessed. The IMine index allows selectively loading into memory only the index blocks used for the local search. Hence, it supports a reduction of disk reads. Since only a small fragment of the data is actually loaded in memory, more memory space is available for the extraction task. Read disk blocks are stored in the buffer cache memory of PostgreSQL.

Table TI_Tree and the I-Btree are accessed by using the read functions available in the PostgreSQL access methods.

3 .ITEM SET MINING

Several algorithms have been proposed for item set extraction. These algorithms are different mainly in the adopted main memory data structures and in the strategy to visit the search space. The IMine index can support all these different extraction strategies. Since the IMine index is a disk resident data structure, the process is structured in two sequential steps: 1) the needed index data is loaded and 2) item set extraction takes place on loaded data. The data access methods presented allow effectively loading the data needed for the current extraction phase. Once data are in memory, the appropriate algorithm for item set extraction can be applied. Frequent item set extraction by means of two representative state-of-the-art approaches, i.e., FP-growth and LCM v.2, is described.

3.1 Frequent Item Set Extraction

We present two approaches, denoted as FP-based and LCM-based algorithms, which are an adaptation of the FP-Growth algorithm and LCM v.2 algorithm, respectively.

FP-based algorithm

The FP-growth algorithm stores the data in a prefix-tree structure called FP-tree. First, it computes item support. Then, for each transaction, it stores in the FP-tree its subset including frequent items. Items are considered one by one. For each item, extraction takes place on the frequent-item projected database, which is generated from the original FP-tree and represented in a FP-tree based structure.

The FP-based algorithm selects frequent items by means of the get_freq_items function. For each item, the corresponding projected database is loaded from the IMine index by means of the Load_Freq_Item_Projected_DB access method. Then, the original FP-growth algorithm is run. With respect to, the FP-based approach reduces memory occupation by loading in memory only the projection exploited in the current extraction phase. Hence, more memory space is available for the extraction process. Data access overhead has been further reduced by exploiting correlation. Items considered in sequence may occur in correlated index paths. When the next item is considered, its paths are already available in memory and do not have to be read again.

LCM-based algorithm

The LCM v.2 algorithm loads in memory the support-based projection of the original database. First, it reads the transactions to count item support. Then, for each transaction, it loads the subset including frequent items. Data are represented in memory by means of an array-based data structure, on which the extraction takes place.

In the LCM-based algorithm, the database projection is read from the IMine index by means of the Load_Support_ Projected_DB access method. Data are stored in the appropriate array-based structure, on which the original LCM v.2 algorithm is run. Since I-Tree paths concisely represent transactions, reading the database projection from the IMine index instead of from the original database is more effective in large databases

.4.PROPOSED RULE MINING SCHEME

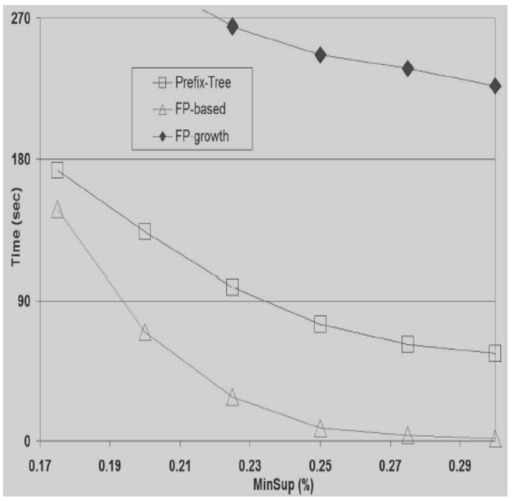
The item set extraction and indexing operations are integrated in the system. The IMine scheme is improved to handle incremental data. Reindexing process is optimized. Data structure is updated to handle all data distribution.

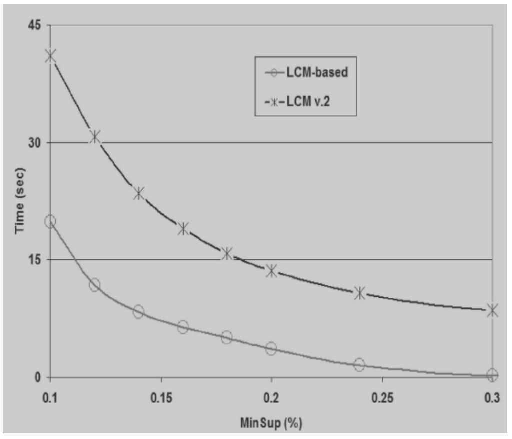
Enhancement of index mine scheme

The association rule mining algorithms uses the candidate set and item sets for the frequent pattern mining process. The minimum support and minimum confidence values are used to select the interested rules. The candidate sets and item sets are prepared for each instance of rule mining process. The attribute analysis and frequency estimation operations are carried out for each rule mining process. All the candidate and item set values are stored in the primary memory only. The index mining model stores all the candidate set and item set values under the database. The indexing task is performed on the candidate set and item set collections.

Support and confidence estimation are performed on the indexed data items. The indexing scheme is designed for the static data sets. All the transaction data values are stored in a database. The attribute analysis, candidate set and item set generation, frequency estimation and indexing operation are done on the fixed data sets. Incremental data sets are not supported by the index mine scheme. The system is designed to enhance the index mine scheme to adapt the indexing and rule mining operations on incremental databases. Transaction table is passed into new data entry, data update and remove operations. Enhanced index mine scheme updates the index with respect to the data value changes. Reindexing is not required in the enhanced index mine scheme. The index is refreshed in the new index mine scheme.

The Following are the Experimental Results:





The system is designed to manage transactional data and indexes. The system also maintains item sets in the database. The indexes are used to extract rules. Transactional data can be managed incrementally. The system is divided into four major modules. They are Item set extraction, Indexing process, Incremental transaction management and Rule mining process.

4.1. Item set Extraction

The item set extraction process is performed for the selected transactional data. Attribute names and its values are used for the item set extraction process. Item sets are build with two or more attribute values. Item sets are physically stored in the database. Transaction list shows the transaction table properties. Column information are extracted in attribute selection module. Attribute name and their data values are used in the item set extraction process. Item set list shows the list of item sets in the transactional data.

4.2. Indexing Process

The indexing process is applied on the extracted item sets. The index maintains the itemsets and associated transaction information. The index also maintains the access information. The index is also stored in the database.

4.3.Incremental Transaction Management

The data add and remove operations are handled in incremental transaction management process. The system updates the transactional data and itemset values. The itemset indexs are also updated. The item set frequency is also updated.

4.4. Rule mining Process

The rule mining process is designed to extract frequent item sets. The support value is used in the rule mining process. The rule mining process has done under the index data values. The rules can be extracted with item set relationships.

5 CONCLUSIONS

The IMine index is a novel index structure that supports efficient item set mining into a relational DBMS. It has been implemented into the PostgreSQL open source DBMS, by exploiting its physical level access methods. The IMine index provides a complete and compact representation of transactional data. It is a general structure that efficiently supports different algorithmic approaches to item set extraction. Selective access of the physical index blocks significantly reduces the I/O costs and efficiently exploits DBMS buffer management strategies. This approach, albeit implemented into a relational DBMS, yields performance better than the state-of-the-art algorithms accessing data on a flat file and is characterized by a linear scalability also for large data sets.

- Provided a complete and compact representation of transactional data
- Supports different algorithmic approaches to item set extraction
- Performance better than the state-of-the-art algorithm
- FP-growth , LCM v.2

6. REFERENCE

[1] M. El-Hajj and O.R. Zaiane, “Inverted Matrix: Efficient Discovery of Frequent Items in Large Datasets in the Context of Interactive Mining,” Proc. Ninth ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (SIGKDD), 2003.

[2] G. Grahne and J. Zhu, “Mining Frequent Itemsets from Secondary Memory,” Proc. IEEE Int’l Conf. Data Mining, 2004.

[3] G. Ramesh, and M. Zaki, “Indexing and Data Access Methods for Database Mining,” Proc. ACM Workshop Data Mining and Knowledge Discovery (DMKD), 2002.

[4] Y.-L. Cheung, “Mining Frequent Itemsets without Support Threshold: With and without Item Constraints,” IEEE Trans. Knowledge and Data Eng., vol. 16, no. 9, pp. 1052-1069, Sept. 2004.

[5] G. Cong and B. Liu, “Speed-Up Iterative Frequent Itemset Mining with Constraint Changes,” Proc. IEEE Int’l Conf. Data Mining, pp. 107-114, 2002.

[6] T. Uno, M. Kiyomi, and H. Arimura, “LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets,” Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations, 2004.

[7] J. Pei, J. Han, and L.V.S. Lakshmanan, “Pushing Convertible Constraints in Frequent Itemset Mining,” Data Mining and Knowledge Discovery, vol. 8, no. 3, pp. 227-252, 2004.

[8] G. Grahne and J. Zhu, “Efficiently Using Prefix-Trees in Mining Frequent Itemsets,” Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations, 2003.