A Study of Various Projected Data Based Pattern Mining Algorithms

Ashish Prajapati[1] Prof. Jigar Patel[2]
[1, 2]Alpha College of Eng. and technology Khatraj, Ahmadabad, India

Abstract—The time required for generating frequent patterns plays an important role. Some algorithms are designed, considering only the time factor. Our study includes depth analysis of algorithms and discusses some problems of generating frequent pattern from the various algorithms. We have explored the unifying feature among the internal working of various mining algorithms. The work yields a detailed analysis of the algorithms to elucidate the performance with standard dataset like Mushroom etc. The comparative study of algorithms includes aspects like different support values, size of transactions.

I. INTRODUCTION

The term data mining or knowledge discovery in database has been adopted for a field of research dealing with the automatic discovery of implicit information or knowledge within the databases. The implicit information within databases, mainly the interesting association relationships among sets of objects that lead to association rules may disclose useful patterns for decision support, financial forecast, marketing policies, even medical diagnosisand many other applications. The development of tools capable in the automatic extraction of knowledge from data. To analyze the huge amount of data thereby exploiting the consumer behavior and make the correct decision leading to competitive edge over rivals[1].

The problem of mining frequent item sets arose first as a sub problem of mining association rules. A priori algorithm is quite successful for market based analysis in which transactions are large but frequent items generated is small in number[2]. Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases such as association rules, correlations, sequences, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems. Also Sequential association rule mining is one of the possible methods to analysis of data used by frequent itemsets3. The original motivation for searching association rules came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Association rules describe how often items are purchased together. For example, associations rule "beer, chips (60%)" states that four out of five customers that bought beer also bought chips. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others[4].

A. *Frequent itemset mining problem*

Studies of Frequent Pattern Mining is acknowledged in the data mining field because of its broad applications in mining association rules, correlations, and graph pattern constraint based on frequent patterns, sequential patterns, and many other data mining tasks. Efficient algorithms for mining frequent patterns are crucial for mining association rules as well as for many other data mining tasks. The major challenge found in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of patterns are generated. Therefore, pruning unimportant patterns can be done effectively in mining process and that becomes one of the main topics in frequent pattern mining. Consequently, the main aim is to optimize the process of finding patterns which should be efficient, scalable and can detect the important patterns which can be used in various ways[5].

II. METHODOLOGY

The methodology used to mine frequent itemsets denoted (Figure1).

A. *FP-Growth Algorithm*

FP-tree algorithm is based upon the recursively divide and conquers strategy; first the set of frequent 1-itemset and their counts is discovered. With start from each frequent pattern, construct the conditional pattern base, then its conditional FP-tree is constructed (which is a prefix tree.). Until the resulting FP-tree is empty, or contains only one single path. (Single path will generate all the combinations of its sub-paths, each of which is a frequent pattern). The items in each transaction are processed in L order. (i.e. items in the set were sorted based on their frequencies in the descending order to form a list)[6]. The detail step is as shown in figure1:

Create root of the tree as a "null". After scanning the database D for finding the 1-itemset then process the each transaction in decreasing order of their frequency. A new branch is created for each transaction with the corresponding support. If same node is encountered in another transaction, just increment the support count by one of the common node. Each item points to the occurrence in the tree using the chain of node-link by maintaining the header table.

After above process mining of the FP-tree will be done by Creating Conditional (sub) pattern bases:

Start from node constructs its conditional pattern base. Then, Construct its conditional FP-tree and FP-Growth Method: Construction of FP-tree

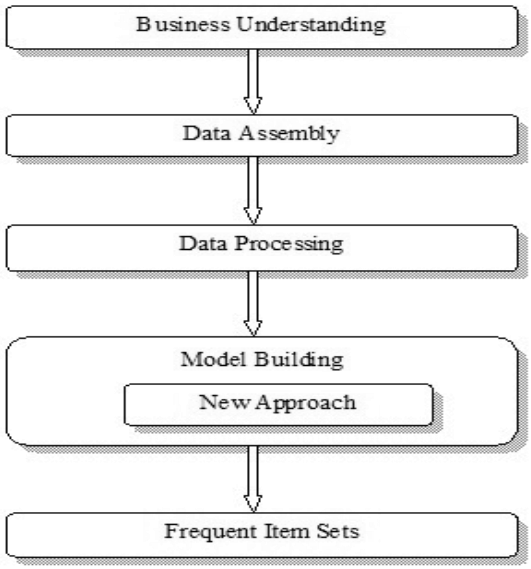

Fig. 1: Methodology used to mine frequent itemsets

Perform mining on such a tree. Join the suffix patterns with a frequent pattern generated from a conditional FP-tree for achieving FP-growth. The union of all frequent patterns found by above step gives the required frequent itemset

In this way frequent patterns are mined from the database using FP-tree.

B. Definition: Conditional pattern base

A "sub database" which consists of the set of prefix paths in the FP-tree co-occurring with suffix pattern.eg for an itemset X, the set of prefix paths of X forms the conditional pattern base of X which co-occurs with X[6].

C. Definition: Conditional FP-tree:

The FP-tree built for the conditional pattern base X is called conditional FP-tree[6].

D. H-mine Algorithm

H-mine algorithm is the improvement over FP-tree algorithm as in H-mine projected database is created using in-memory pointers[7]. H-mine uses an H-struct new data structure for mining purpose known as hyperlinked structure. It is used upon the dynamic adjustment of pointers which helps to maintain the processed projected tree in main memory therefore H-mine proposed for frequent pattern data mining for data sets that can fit into main memory. It has polynomial space complexity therefore more space efficient then FP-Growth and also designed for fast mining purpose. For the large databases, first in partition the database then mine each partition in main memory using H-struct then consolidating global frequent pattern[7]. If the database is dense then it integrates with FP- Growth dynamically by detecting the swapping condition and constructing the FP-tree.

This working ensures that it is scalable for both large and medium size databases and for both sparse and dense datasets. The advantage of using in-memory pointers is that their projected database does not need any memory, the memory required only for the set of in-memory pointers.

E. Recursive Elimination (RELIM)

A close relative of this approach is the H-mine algorithm [8]. In a preprocessing step delete all items from the transactions that are not frequent individually, i.e., do not appear in a user-specified minimum number of transactions. This preprocessing is demonstrated, which shows an example transaction database on the left (table 1). The frequencies of the items in this database, sorted ascending, are shown in the table in the middle. If we are given a user specified minimal support of 3 transactions, items f and g can be discarded after doing so and sorting the items in each transaction ascending w.r.t. Their frequencies we obtain the reduced database shown on the right (table 1).

a d f	g 1 f 2	a d
c d e		e c d
b d		b d
a b c d	e 3	a c b d
b c	a 4	c b
a b d	c 5	a b d
b d e	b 7	e b d
b c e g	d 8	e c b
c d f		c d
a b d		a b d

Table. 1:Count for Experiment

Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted ascending w.r.t. their frequency (right)

Then select all transactions that contain the least frequent item (least frequent among those that are frequent), delete this item from them, and recurs to process the obtained reduced.

Database, remembering that the item sets found in the recursion share the item as a prefix. On return, remove the processed item also from the database of all transactions and start over, i.e., process the second frequent item etc. This process is illustrated for the root level of the recursion, which shows the transaction list representation of the initial database at the very top (figure 2). In the first step all item sets containing the item e are found by processing the leftmost list. The elements of this list are reassigned to the lists to the right (grey list elements) and copies are inserted into a second list array (shown on the right). This second list array is then processed recursively, before proceeding to the next list, i.e., the one for item a.

In these processing steps the prefix tree (or the H-struct), which is enhanced by links between the branches, is exploited to quickly find the transactions containing a given item and also to remove this item from the transactions after it has been processed.

F. Split and Merge (SaM) algorithm

The Split and Merge (SaM) algorithm is a simplification of the already fairly simple Recursive Elimination (RElim) algorithm[9]. The split and merge algorithm employs only a single transaction list

(purely horizontal representation), stored as an array. The steps are illustrated for a simple example transaction database: step 1 shows the transaction database in its original form (figure 3). In step 2 the item frequencies are determined in order to discard infrequent items. With a minimum support of 3, items f and g are infrequent and thus eliminated. In step 3 the (frequent) items in each transaction are sorted according to their frequency, because processing the items in the order of increasing frequency usually leads to the shortest execution times. In step 4 the transactions are sorted lexicographically into descending order, with an item with higher frequency preceding an item with lower frequency. In step 5 the basic data structure is built by combining equal transactions and setting up an array, in which each element consists of two fields: an occurrence counter and a pointer to the sorted transaction.

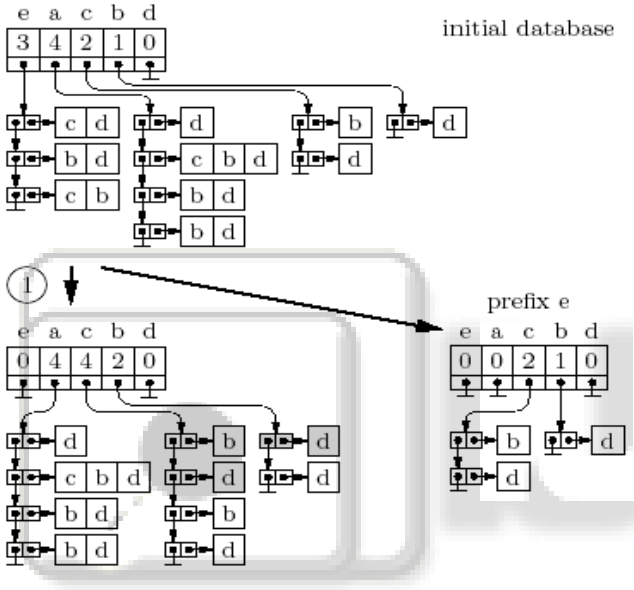

Fig. 3: Procedure of the recursive elimination with the modification of the transaction lists (left)
As well as the construction of the transaction lists for the recursion (right)

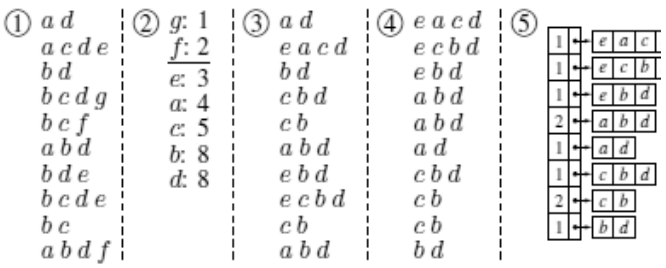

Fig. 4: An example database: original form (1), item frequencies (2), and transactions with sorted items (3), lexicographically sorted transactions (4), and the used data structure (5)

The basic operations of the recursive processing, which follows the general divide-and-conquer scheme are illustrated in the split step (left) the given array is split w.r.t. the leading item of the first transaction (item e in our example).

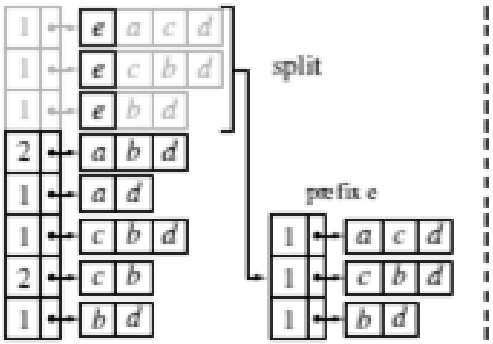

Fig. 4: The basic operations split

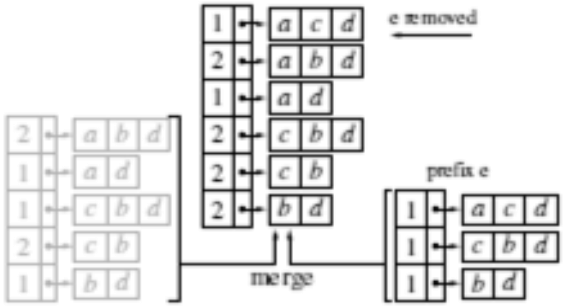

Fig. 5: The basic operations: merge

All elements referring to transactions starting with this item are transferred to a new array (figure 4). In this process the pointer (in) to the transaction is advanced by one item, so that the common leading item is "removed" from all transactions. Obviously, this new array represents the conditional database of the first sub problem, which is then processed recursively to find all frequent item sets containing the split item (provided this item is frequent).

The conditional database for frequent item sets not containing this item is obtained with a simple merge step in right part (figure 4). The new array and the rest of the original array are combined with a procedure that is almost identical to one phase of the well-known mergesort algorithm. Since both arrays are lexicographically sorted, one merging traversal suffices to create a lexicographically sorted merged array. The only difference to a mergesort phase is that equal transactions (or transaction suffixes) are combined: There is always only one instance of each transaction (suffix), while its number of occurrences is kept in a counter. In our example this results in the merged array having two elements less than the input arrays together: the transaction (suffixes) cbd and bd, which occur in both arrays, are combined and their occurrence counters are increased to 2.

III. RESULTS AND DISCUSSION

Data set: The dataset was obtained from the UCI repository of machine learning databases[10]. The characteristics of Mushroom dataset selected for the experiment (table 2).

File name	Number of Records	Number of Columns
Mushroom.D90.N8124.C2.num	8124	23

Table. 2: Characteristics of Mushroom Dataset

IV. RESULT ANALYSIS

We have conducted a detailed study to assess the performance of projected data based frequent pattern mining algorithms. The performance metrics in the experiments is the total execution time taken and the number of patterns generated for Mushroom data set. For this comparison also same data sets were selected as for the above experiment with 30% to 70% of minimum support threshold.

The execution time for all the algorithms with different support threshold for Mushroom data set (table 3). The time o f execution is decreased with the increase support threshold. The execution time of the all discussed algorithm is nearby but it can also be analyzed that the execution time of SaM is comparatively less for higher support threshold.

Support	Total Execution Time in Seconds			
	FP-Growth	H-mine	Relim	SaM
30	0.13	0.11	0.11	0.11
40	0.11	0.11	0.09	0.10
50	0.09	0.09	0.09	0.08
60	0.08	0.09	0.08	0.07
70	0.08	0.08	0.07	0.06

Table. 3: Total execution time using mushroom dataset

V. CONCLUSION

In this work, an in-depth analysis of few algorithms is done which made a significant contribution to the search of improving the efficiency of frequent pattern mining. By comparing them to classical frequent pattern mining algorithms strength and weaknesses of these algorithms were analyzed.

A comparison framework has developed to allow the flexible comparison of FP-growth, H-mine, Relim and SaM algorithms. The execution time of the all discussed algorithm is nearby but it can also be analyzed that the execution time of SaM is comparatively less for higher support threshold.

REFERENCES

[1] Raorane A.A., Kulkarni R.V. and Jitkar B.D., Association Rule – Extracting Knowledge Using Market Basket Analysis, Res. J. Recent Sci., 1(2), 19-27 (2012)

[2] Agrawal R. and Srikant.R, Fast algorithms for mining association rules, In Proc. Int'l Conf. Very Large Data Bases (VLDB), 487–499 (1994)

[3] Shrivastava Neeraj and Lodhi Singh Swati, Overview of Non-redundant Association Rule Mining, Res. J. Recent Sci., 1(2), 108-112 (2012)

[4] Agrawal R., Imielienski T. and Swami A., Mining Association Rules between Sets of Items in Large Databases, Proc. Conf. on Management of Data, 207–216 (1993)

[5] Pramod S., Vyas O.P., Survey on Frequent Item set Mining Algorithms, In Proc. International Journal of Computer Applications (0975 - 8887), 1(15), 86–91 (2010)

[6] Han J., Pei H. and Yin. Y., Mining Frequent Patterns without Candidate Generation, In Proc. Conf. on the Management of Data (2000)

[7] Pei. J., Han. J., Lu. H., Nishio. S. Tang. S. and Yang. D., H-mine: Hyper-structure mining of frequent patterns in large databases, In Proc. Int'l Conf. Data Mining (2001)

[8] Borgelt C., Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination, Proc. Workshop Open Software for Data Mining (OSDM'05 at KDD'05, Chicago, IL), 66–70 (2005)

[9] Borgelt C., SaM., Simple Algorithms for Frequent Item Set Mining, IFSA/EUSFLAT 2009 conference (2009)

