PLT- Positional Lexicographic Tree: A New Structure for Mining Frequent Itemsets

Azzedine Boukerche and Samer Samarah
School of Information Technology & Engineering
University of Ottawa, Ottawa, Canada
{boukerch, ssamarah@site.uottawa.ca}

Abstract

Association rules have proved their influence in different industrial fields, where their goal is to identify the relations existing among the events that are stored in large databases. However, in order to enumerate the association rules, there is a need to identify the frequent set of itemsets (i.e. those events that occur together in a sufficient number of transactions). In this paper, a new representation structure for the data stored in any transactional database is proposed. This structure, which we refer to as Positional Lexicographic Tree (PLT), provides an efficient mechanism for subset checking based on a summary of the data extracted from the database. This makes PLT a promising tool for most of the existing data mining approaches. Moreover, our proposed PLT structure regulates the data in the database so that they can be applicable to compression and indexing techniques, which makes PLT suitable for supporting large databases. First, we introduce the PLT construction process, then highlight the different mining approaches that can be modulated to take advantage of PLT. We then present our algorithm and finally prove its correctness.

1. Introduction

Recent developments in information technology have allowed a massive amount of data to be generated and stored in large databases. These enormous databases create a challenge in terms of discovering hidden patterns presented in the data; these patterns cannot be discovered using the traditional query languages associated with the current database management systems, and are important in that they can improve the decision-making process.

Data mining has received a great deal of attention in recent years within the database community, where different data mining techniques have been proposed to extract hidden patterns from large databases. Among many interesting domains of data mining, transactional databases of supermarkets have been the focus of many researchers, where each record of these databases contains information such as the transaction identifier, the date of the transaction, the set of items bought in that particular transaction, and possibly the customer identifier. A data mining problem in this context is mining the association rules; this problem was first introduced in [1]. These rules give an idea of the set of items that can be bought within the same transaction. An example of such rules might be that 95 % of customers who buy item *X* are willing to buy item *Y* in the same transaction (95% is the support of the rule). Generating all the association rules will help mangers make decisions about their business, such as which items should be placed next to or near each other, catalog design, customers' buying habits, and butting items on sale. In addition, association rules have been applied to other domains such as medical data and web page access habits.

Theoretically, if we have a set of n events (items), there is a potential for 2^n relations to exist among these events; these relations are used to formulate the association rules. In order to compute the support of the rules, the set of relations should be checked against the database to determine the frequency of each relation; for large n, this huge search space will degrade the performance of any algorithm designed to solve this problem. Another dimension that adds more computational costs to the overall process is the large size of the database, which must be scanned several times in order to generate the frequencies. All of the algorithms in the literature tried to utilize different heuristics and pruning techniques in order to efficiently determine the set frequent relations (a relation is frequent if its frequency is greater than the user's predefined threshold).

The problem of the association rules can be divided into two steps [1]. In the first step, the set of itemsets (itemset and relation refer to the same concept) that exceeds a predefined threshold are determined; these itemsets are called frequent. In the second step, the association rules are determined from this set of frequent itemsets. In this paper, we introduce a new representation for the data stored in the database; this new representation takes the form of a lexicographic

tree that will be used to enumerate the frequent relations present in large databases. Our enhancement is in the way in which this representation makes subset checking a light process, as well as the applicability of this data to compression and indexing techniques.

This paper is organized as follows. Section 2 provides a formula definition for frequent itemset generation. Section 3 surveys some of the existing methods. Section 4 defines our new representation for the data (the positional lexicographic tree). Section 5 illustrates the construction and mining procedure. Section 6 concludes the paper.

2. Problem Definition

As defined in [2], the problem of mining association rules can be formulated as follows: Let $I = \{i_1, i_2,...,i_n\}$ be a set of distinct items. Let D be a set of transactions where each transaction T is a set of items such that $T \subseteq I$, each transaction $T \in D$ has an associated identifier uniquely identifying it called *TID*. Let X be a subset of items such that $X \subseteq I$, also referred to as an itemset. A subset with K items is called K-itemset. A transaction T supports X if $X \subseteq T$. X has a support s, if $s\%$ of the transactions in D support X[1]. The association rules are implications of the form $X \Rightarrow Y$ where $X \subset I$, $Y \subset I$ and $X \cap Y = \Phi$

The support of the rule ($X \Rightarrow Y$) is the support of the itemset ($X \cup Y$). We say that the rule has a confidence c if $c\%$ of the transactions that contain X also contain Y.

The Association rules mining problem over database D, is to find all the association rules that have support and confidence greater than or equal to the user predefined minimum support and minimum confidence respectively.

Once the frequent itemset are determined, generating the rules is straightforward; so the focus of the paper will be on how to enumerate the frequent itemsets efficiently.

3. Related Works

Several algorithms in the literature proposed to attack the problem of mining frequent itemsets from large databases; these algorithms differ mainly in the way in which they represent the database and in the way in which they generate the frequent itemsets. These algorithms can be classified into two main approaches: the candidate generation approach and the pattern growth approach. In terms of these approaches, the algorithms also differ in the way in which they represent the database. The two most popular formats are vertical layout, where each item is associated with the list of transaction identifiers where it was occurred, and horizontal layout, where each transaction identifier is associated with the list of items.

The candidate generation approach enumerates the frequent itemsets in a level wise manner, with several scans for the database. In each iteration, the itemsets found to be frequent are used to generate the candidate (possible frequent itemsets) to be counted in the next iteration. Within this approach are the AIS algorithm [1], Apriori, AprioriId, and AprioriHyprid [2], DHP (Directed Hashing and Pruning) [5], the Partition algorithm [6], and DIC (Dynamic Itemset Counting) [7]. The most popular algorithm of the candidate generation approach is the Apriori. Others except AIS are optimization of Apriori.

In the Apriori algorithm, a database scan is conducted in order to determine the set of frequent items. From this set of items it then generates a candidate set to be counted in the next step by joining the set of frequent itemset found to be frequent in the current pass. In addition, it prunes the set of candidate based on the anti monotone property, which states that all subsets of frequent itemsets should be frequent in order to eliminate candidates that have at least one infrequent subset. This process is repeated a number of times equal to the size of the largest frequent itemset.

The pattern growth approach tries to avoid the large number of candidates generated in each pass and overcome the repeated scans of the database, which makes most of the algorithms in this approach to outperform most of the candidate generation approach algorithms.

The core algorithm of the pattern growth approach is the Frequent Pattern growth (FP-growth) proposed by Han et al, [3]. In this method, the database is converted to a compact representation in the form of a tree called Frequent Pattern tree (FP-tree), which is much smaller in size than the original database. The FP-tree is constructed in such a way that all relevant information needed in the mining process is presented in the tree structure. Building the tree structure requires only two scans for the database. After building the FP-tree, the FP-growth routine mines all the frequent patterns from the tree structure without referring to the original database and without generating candidates. Only one itemset can be considered at a time, and a new tree is constructed from the set of itemsets that occur. This tree is called conditional structure; this

[1] Through this paper we will refer to the support as the number of transactions that contain the itemset rather than ratio.

process continues recursively until all the frequent itemsets are generated.

Although the FP-growth method has proven its efficiency in comparison to the candidate generation approach, [8] noticed that the FP-growth method is not suitable for all kinds of data. Specially when the database is sparse, the resulting FP-tree is very large and there is significant overhead in traversing the FP-tree in the mining process. This led to the need for a huge amount of space in the recursive process, which will prevent the FP-growth method to scale well for large amount of data. Several algorithms, such as (H-Mine [8], FP-growth*, [4], COFI-tree [10], and ITI-tree [19], CT-ITL [13]), have been proposed to overcome these limitations

4. The PLT Structure Model

In this paper, a new annotation for the lexicographic tree is introduced. This new presentation allows data to be stored in a compressed form and provides an easy mechanism to move between the elements of the tree and an easy way for subset checking during the mining process.

Let $I = \{i_1, i_2, ...i_n\}$ be a set of items. A lexicographic order is assumed to exist among the items of I; a lexicographic prefix tree is a tree representation where the root node is labeled with null and all other nodes represent elements in the set I listed in lexicographic order from left to right. Each node is linked to the nodes that represent the items that occur after it in the lexicographic order. Figure 1 illustrates the lexicographic tree of the set {A, B, C, D}.

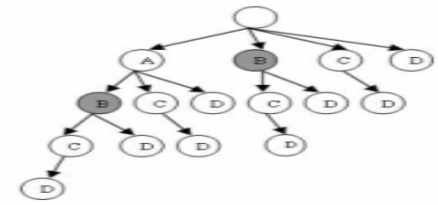

Figure 1. The lexicographic tree of items {A, B, C, D}

4.1. The Positional Lexicographic Tree

Definition 4.1.1: *Rank (i)* is a function that maps each element ($i \in I$) to a unique integer in such a way that the lexicographic order is maintained.

Definition 4.1.2: *pos(n)* is a function that maps each node in the lexicographic tree to an integer that represents its position among its siblings to the parent node starting from left to right.

For example, in Figure 1, node C is a child of node A at level 2 and $pos(C) = 2$ (i.e. C is in the position of two lexicographically as a child of A).

Definition 4.1.3: Let X_k be a node at level k. $V(X_k) = [pos(x_1), pos(x_2),.....,pos(x_k)]$ is then a position vector that encodes the position values of the nodes that formulate the path from the root to node X_k.

Lemma 4.1.1: Let $X = \{x_1, x_2,.....,x_k\}$ be an itemset and let $V(X) = [pos(x_1), pos(x_2),.....,pos(x_k)]$ be the corresponding position vector of X . For each x_i in X, it then holds

$$Rank(x_i) = \sum_{j=1}^{i} pos(x_j)$$

During the construction process, the Rank function is used to calculate the positions of the nodes as follows: $pos\ (j) = Rank\ (j) - Rank\ (i)$ where j is a child node of node i and $Rank(null) = 0$. Figure 2 represents the Positional Lexicographic Tree (the new compressed structure) for set {A,B,C,D}. Each node in Figure 2 is associated with the integer (position value) that represents its position for the parent node.

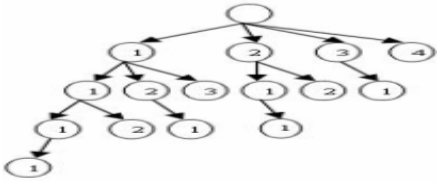

Figure 2. The PLT structure

Lemma 4.1.2: For each subset $X \in P(I)$, $P(I)$ is the power set of the set I, $V(X)$ uniquely determines itemset X in the positional lexicographic tree.

Proof: Let $X = \{x_1, x_2,.....,x_k\}$, $Y = \{y_1, y_2,.....,y_m\}$ be two itemsets such that $X \neq Y$.

Case 1: X, Y are of a different size:

$V(X) \neq V(Y)$ because of characteristic of vector equality (two vectors must be of the same size in order to be equal).

Case 2*:* X, Y are of the same size ($k = m$):

Assume that $V(X) = V(Y)$

➔ $[pos(x_1),pos(x_2),..,pos(x_k)] = [pos(y_1), pos(y_2),..,pos(y_m)]$.

➔ $pos(x_i) = pos(y_i)$ $1 <= i <= k$.

➔ $Rank(x_i)\text{-}Rank(x_{i-1}) = Rank(y_i)\text{ -}Rank(y_{i-1})$, $1 < i <= k$.

Assume that $x_i = y_i$ for $1<i <= k$

➔ $Rank(y_{i-1}) = Rank(x_{i-1})$ $1< i <= k$

By definition of the Rank function (the uniqueness property), these sets of equalities can not be held unless we have $X = Y$, which is a contradiction. □

Property 4.1.1: Let T_{ik} be a sub-tree rooted with item i at level k in the lexicographic tree. Let j be the parent of node i. T_{ik} then has n-repeated structures (sub-trees) at level $(k+1)$ within the same sub-tree rooted with node j, where n is the number of siblings that proceed item i in the lexicographic tree according to parent j.

Referring once again to Figure 1, the sub-tree rooted with node B in level 1 has the same structure under its left sibling (node A) at level 2 (the sub trees rooted with the gray nodes).

Lemma4.1.3: Let $V(X) = [pos(x_1), pos(x_2),\ldots\ldots,pos(x_k)]$ be a position vector of size k. The position vector representation of any "*k-1*" level subset of itemset X then has one of the following forms:

a) $[pos(x_1), pos(x_2),\ldots\ldots,pos(x_{k-1})]$.

or

b)$[pos(x_1),pos(x_2),\ldots,pos(x_i)+pos(x_{i+1})\ldots,pos(x_k]$ for a given $1<= i <k$. That is, the position vector of a potential subset of X at level $k-1$ is achieved by replacing two consecutive positions in $V(X)$ with their sum.

Proof: Let $X =\{x_1, x_2,\ldots\ldots,x_k\}$ and $V(X) = [pos(x_1), pos(x_2),\ldots\ldots,pos(x_k)]$.

Known: Any subset of X at level $k-1$ is any itemset $X`$ that is produced by removing an element from X one at a time.

- Assume that the missing element is element x_k. Thus, $X`=\{x_1,x_2,..,x_{k-1}\}$ and $V(X`) = [pos(x_1),pos(x_2),\ldots,pos(x_{k-1})]$…….lemma 4.1.3.a.

- Assume that the missing element is x_i, where $1 <= i < k$:

➔$X' = \{x_1,x_2,x'_i, x_{i+2},..,x_k\}$ 1

We claim that $pos`(x'_i)$ in $V(X')$ is equal to $pos(x_i) + pos(x_{i+1})$ in $V(X)$;

➔ $pos(x'_i) = pos(x_i) + pos(x_{i+1})$

➔$Rank(x'_i) – Rank(x_{i-1}) = pos(x_i) + pos(x_{i+1})$

➔$Rank(x'_i) = pos(x_i) + pos(x_{i+1}) + Rank(x_{i-1})$……..2

From 1, we have

$pos`(x'_i) = Rank(x'_i) – Rank(x_{i-1})$.

Replace $Rank(x'_i)$ by its definition from 2 we have

$pos`(x'_i) = pos(x_i) + pos(x_{i+1}) +Rank(x_{i-1}) – Rank(x_{i-1})$.

$= pos(x_i) + pos(x_{i+1})$ our claim.

4.2. The Positional Lexicographic Tree

Now the complete process of constructing the PLT can be represented. We begin by providing an illustrative example followed by a formula description of the algorithm.

Table 1 shows a database of six transactions. In the first step, the database is scanned to determine the set of frequent 1-itemset. Since the only frequent items can participate in formulating the frequent itemsets, this step aims to eliminate those items that less support than the user's predefined minimum support. Assume that the absolute support count is 2. The set of frequent 1 items are then{(A,4),(B,5),(C,5),(D,4)}. The number beside the item refer to its frequency. The next step is to associate a unique number with each item using the Rank function. As a result we have: Rank(A) =1, Rank(B) = 2, Rank(C) = 3, Rank(D) = 4.

Table 1. Transactional database

TID	Item Set
1	ABC
2	ABC
3	ABCD
4	ABDE
5	BCD
6	CDF

In the second step, another scan of the database is conducted; for each transaction, the set of infrequent items is filtered out and a positional vector is created and inserted in a table according to its length (we assume that a table-like data structure is used to represent the positional tree; a physical tree may also be assumed). If this vector previously existed we merely increment its frequency. Otherwise, we simply add it with a support count equal to 1. In other words, we partition the database to a set of partitions such that each partition stores the vectors of the same length. In addition, we store the summation of the position values presented in the vector with each vector. This value will be used during the mining procedure using the conditional approach. Figure 3 provides two different perspective for the data using the tree structure and the matrix structure. Algorithm 1 gives the formula specification for the positional tree construction process.

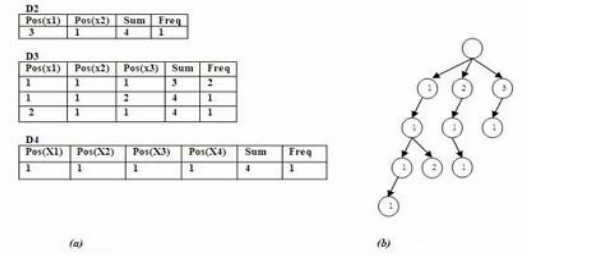

Figure 3. (a) The matrices structure (b) The tree structure

Algorithm 1: PLT Construction

Input: D, min_sup
Output: PLT

Generate frequent 1 items.
For each transaction t ∈ D
Let t' = [t'$_1$,t'$_2$,..,t'$_k$] the frequent items in t.
Generate V(t')

V(t').sum = $\sum_{i=1}^{k} pos(t'_i)$

If V(t') ∈ D_k
Increment the current frequency
Else
Add V(t') with V(t').Freq =1

5. The PLT Mining Process

It is clear that the support count of any itemset is equal to the number of times it occurs as a single transaction plus the number of transactions in which occurs as a subset. The next two subsections discuss how the positional lexicographic tree can be used during the mining process using two different approaches. Top down and conditional.

5.1. The Top down Approach

In this methodology, we start with the database partition that represents the itemset of maximum length. For example *k*. Then for each entry presented in D_k of the form [p_1,p_2,….,p_k] with ps as positional values, we generate the vectors of the first two subsets at level k-1 as follows:

A) The first subset is constructed based on lemma 4.1.3.a by removing the last positional value (p_k)
B) The second subset is constructed by replacing (p_k and p_{k-1}) by their sum, lemma 4.1.3.b.

For reasons of efficiency and correctness, we may include the first step above in the positional tree construction process since we need this step only for the already existed vectors not for those that are generated using part B above. This is done by adding the vector at different levels in the database by considering one position at a time. For instance, vector [1,1,1,1] should be added as [1,1,1], [1,1] at D_3 and D_2 partitions. In this case, the top down approach is started by considering part B above and constructing the vector [p_1,p_2,….,p_{k-1}+p_k] and adding it to the D_{k-1} partition. The same thing is done for all the vectors in D_k and the vectors in the other partitions by considering the last tow positions. After this, part B from above is repeated, but this time the process occurs by shifting one position to the left (i.e. the pair (p_{k-2}, p_{k-1}) for level k). The new vector is [p1,p2,..,p_{k-1}+p_{k-2},p_k), and the same is the case for all other partitions one shift to the left from the pervious location considered; any vector that does not have enough space for shifting has already gone through the mining process. The entire process will continue until there is no longer enough space to shift in any vector. The same condition should be applied to the vectors that were generated in previous iteration if they can satisfy the shifting (i.e. consecutive positional can be added). In this way, all of the subsets are generated and frequency support is inherited by the subsets without duplications.

Pos(x_1)	Pos(x_2)	Freq
1	1	4
2	1	4
3	1	3
1	2	3
1	3	2
2	2	3

Pos(x_1)	Pos(x_2)	Pos(x_3)	Freq
1	1	1	3
1	1	2	2
2	1	1	2
1	2	1	1

Pos(X_1)	Pos(X_2)	Pos(X_3)	Pos(X_4)	Freq
1	1	1	1	1

Figure 4. The database after top-down approach

At the end of the above procedure, the database contains all the frequencies of all the subsets that may be presented in the database, and the frequent items are those that have frequency grater than the predefined minimum support. Figure 4 shows the database after the above procedure has been applied to the database in Table 1. We assumed that part A was constructed during the constructions process. Algorithm 2 presents the formula specification for part B.

Algorithm 2: Top Down Approach

Input: PLT Structure
Output: All Subsets with their frequencies

k = Maximum Vector Length.
For j = k down to 2
L=j
For i = k down to 2
For each V ∈ D_i
If p_L exist in V
p' = p_L + p_{L-1}
V' = V | p' replaced by p_L,p_{L-1}
If V' ∈ D_{i-1} then
V'.Freq += V.Freq
Else
Add V' with V'.Freq = V.Freq
L= L-1

5.1. The Conditional Approach

The last subsection showed how the top down approach can be used in the mining process. Although this approach provides an efficient way to propagate the frequency from an itemset to all of its subsets, it does not fully utilize the anti-monotone property: "all subsets of frequent itemsets should be frequent". Later we will discuss the appropriate situation for the top down methodology. Here we will discuss the possibility of having the positional tree cooperate with the pattern growth approach that uses conditional intermediate data structures. In this approach, the anti-monotone property can be fully utilized; the contribution of the positional lexicographic tree in this method is the way in which it provides subset checking between itemsets.

In the conditional method, the process begin be considering an (item / itemset) at a time and constructing its conditional database [3]. The conditional database of an itemset provides a model for all of the supersets of the itemset presented in the original database. From this conditional database an extension of the itemset then made from the frequent items presented in its conditional database, and the support count is computed from this database. A new conditional database is constructed as long the produced itemset is frequent. If the new extension is no longer frequent, there is no need for a new conditional database and the mining process ends with this itemset; if it is frequent, another extension is considered and the process continues recursively. The contribution of positional lexicographic tree in this approach is in the way in which the conditional tree of the new extended itemset is identified, using lemma 4.1.1, and in the way how the conditional structure presented where we assume it as a new PLT structure and called it conditional PLT.

For the database in Table 1, starting from the highest ranked item, which is 4 (i.e. we are in the process of building the conditional database of item D), the conditional database for item D is the database that contains vectors with a sum equal to D's rank. The support frequency of D is equal to the sum of the frequencies of the vectors that participate in building D's conditional database. In order to obtain the correct result, during the process of constructing D's database from the original database, for each vector support D a new vector is constructed by removing the last position value and inserting this vector into the proper partition in the original database. Based on this discussion, Figure 5 (a) and (b) shows the original database after extracting D's conditional structure and D's conditional structure itself. The process continues by considering D's conditional database and considering extensions for D one by one in reverse lexicographic order and building the conditional database of (D $\cup$ the new extension) where the new extension is the highest ranked item presented in D's conditional PLT. If the new support is frequent, the mining process continues. Otherwise, the mining process ends at this stage. Algorithm 3 provides a formula description for this conditional mining approach using PLT, which is first invoked with an empty set as an itemset and the PLT.

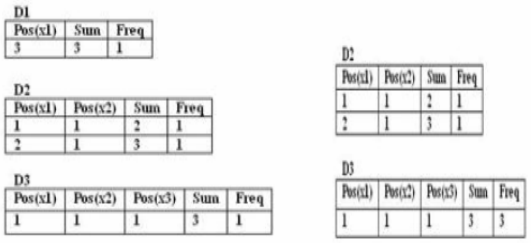

Figure 5. (a) D's conditional database (b) The PLT

6. Conclusion

In this paper, a new data structure for mining association rules was proposed. We successfully showed how this structure can be used with the two most popular mining methods, the top down and conditional approaches. Compared to pervious representations of the data, PLT provides a promising structure that is applicable to indexing and compression techniques, which makes PLT a solution when large databases are being mined. In additions, PLT is considered to be self-continued structure, which means that there is no need for any other data structure during the mining process. This is different from pervious methods such as the Apriori algorithm that required several data structures during the mining process. Another contribution of PLT is the way in which it provides subset checking which, is considered to be one of the heaviest steps in the mining process , and the way in which it identifies the conditional structure, which is easier in PLT compared to the links presented in the FP-tree. PLT provides partition criteria that makes it easy to partition the mining process into several separate tasks; each can be accomplished separately. We have also shown how the top down approach does not employ the anti-monotone property, which makes it suitable for situations where a very low minimum support is provided. Or, if it

coupled with a strategy with which to compute the frequency and high level, the conditional approach is best used when the data is dense and a high support count is required.

Algorithm 3: Conditional Approach

Input: PLT Structure
Output: Frequent itemsets

Conditional_Construct(PLT, j)
For each V=[p_1,..,p_k] in D with V.Sum = j
  V' = [p_1,..,p_{k-1}]
  Update PLT with V'
  Add V' to CD_j
  CPLT = PLT_Construction(CDj, Min_Supp)
Return CPLT

Mining (PLT, itemset)
Max = Maximum Rank in PLT
tem = itemset
For j = Max down to 1
  CPLT = Conditional_Construct(PLT, j)
  itemset = itemset $\cup$ j
  Output itemset
  Mining(CPLT, itemset)
  itemset = tem.

References

[1] R. Agrawal, T. Imilienski, A. Swami. Mining association rules between sets of items in large databases. In proceedings of the ACM SIGMOD international conference on Management of data, PP: 207-216, 1993.Panther, J. G., Digital Communications, 3rd ed., Addison-Wesley, San Francisco, CA (1999).

[2] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. Proc. of the 20th VLDB Conf.,, 1994.

[3] J. Han, J. Pei, Y. Yin. Mining Frequent Pattern without Candidate Generation. In Proceedings of the 2000 ACM SIGMOD international conference on Management of data, PP: 1-12, Dallas, Texas,USA, May 2000.

[4] G. Grahne, J. Zhu. Efficiently Using the Prefix-trees in Mining Frequent Itemsets. FIMI'03, Workshop on Frequent Itemset Mining Implementations, Melbourne, Florida, USA, November 2003.

[5] J.S. Park, M. Chen, P.S. Yu. An Effective Hash-Based Algorithm for Mining Association Rules. In Proceedings of the ACM SIGMOD International Conference on the Management of Data, pages 175-186, May 1995.

[6] S. Brin, R. Motwani, J.D. Ulman, S. Tsur. Dynamic Itemset Counting and Implication Rules for Market Basket Data. In Proc. of the ACM SIGMOD international conference on Management of data, PP: 255-264, 1997.

[7] J. Pei et al. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. In proceeding of the IEEE Conference on Data Mining (ICDM), San Jose, California, USA, November 2001.

[8] R. Agarwal, C.Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent itemsets. Journal Parallel and Distributed Computing, 2000.

[9] R. Agarwal, C.Aggarwal, and V. Prasad. A tree projection algorithm for generation of frequent itemsets. Journal Parallel and Distributed Computing, 2000.

[10] M. EL-Hajj, O.R Zaiane. Non Recursive Generation of Frequent K-itemset from Frequent Pattern Tree Representation. FIMI'03, Workshop on Frequent Itemset Mining Implementations, Melbourne, 2003.

[11] M.J. Zaki. Parallel and Distributed Association Mining: A survey. IEEE Concurrency. Vol.7(4), pp.14-25, 1999.

[12] M.J. Zaki. Scalable Algorithms for Association Mining. IEEE Trans. On Knowledge and Data Eng., 12 (3) 2000

[13] Y.G. Sucahyo, R.P. Gopalan. CT-ITL: Efficient Frequent Item Set Mining Using a Compressed Prefix Tree with Pattern Growth. In proceeding of the 14th Australasian Database Conference, Vol 17, 2003..

[14] R. Agrawal, J. C. Shafer. Parallel Mining of Association Rules, IEEE Transactions on Knowledge and Data Engineering Vol8 , No 6, PP. 962 – 969, 1996 .

[15] EH. Han, G. Karypis, V. Kumar. Scalable parallel data mining for association rules, In proceeding of the ACM SIGMOD international conference on Management of data, PP. 277-288, Tucson, Arizona, USA, 1997.

[16] M.J. Zaki, K. Gouda. Fast Vertical Mining using Diffsets. In proceeding of the 9th International Conference on Knowledge Discovery and Data Mining, Washington, DC, August 2003.

[17] D. Kreher, D. Stinson. Combinatorial Algorithms: generation, enumeration and search. CRC Press. 1998.

[18] S.J. Yen, A.L.P. An Efficient Approach to Discovering Knowledge from Large Databases. In Proceedings of the fourth international conference on Parallel and distributed information systems. PP:8-18.. 1996.

[19] R.P. Gopalan, Y.G. Sucahyo. Tree ITL-Mine: Mining Frequent Itemsets Using Pattern Growth, Tid Intersection and Prefix Tree. In proceeding of the 15th