A NEW SET ENUMERATION FOR MINING FREQUENT ITEMSETS IN GENERALIZED ASSOCIATION RULE MINING

*Thanaruk Theeramunkong and **Kritsada Sriphaew

Information Technology Program
Sirindhorn International Institute of Technology
Thammasat University
P.O. Box 22 Thammasat Rangsit Post Office, Pathumthani 12121, Thailand
Phone:+66-2-986-9103(-8) Ext.2004, Fax:+66-2-986-9112(-3)
Email: *ping@siit.tu.ac.th **kong@siit.tu.ac.th

ABSTRACT

Generalized association rule mining is a generalization of association rules mining based on taxonomy, to discover more informative knowledge from database. Previous approaches applied traditional set enumeration, that needs a computational expensive process to generate candidate itemsets and check whether they are frequent or not. In this paper, we propose a new set enumeration without such intensive checking process.

1. INTRODUCTION

In the area of knowledge discovery in database, association rule mining is an important task, firstly introduced in [1], to find the set of all subsets of items (called itemsets) that frequently occur in database records (transactions), and to extract the rules indicating how a subset of items influences the presence of another subset [2]. However, finding association rules sometimes encounters difficulties of finding desired knowledge in database. For example, we may obtain the rule "5% of customers who buy wheat bread, also buy chocolate milk" which has more specific knowledge and less probability than the general rule "30% of customers who buy bread also buy milk". The latter rule is called generalized association rule. It has more informative, initiative and flexible than the former one.

Unlike traditional association rule mining, generalized association rule mining includes so called generalized items which exist in a *taxonomy* (is-a hierarchy) over the items. For example, a taxonomy for market basket data may classify items (or products) into brands, categories, product groups, etc. Similar to that of association rules mining [1], the task of generalized association rule mining consists of two steps: (1) finding all frequent itemsets, and (2) generating all high confidence rules. These frequent itemsets and high confidence rules will also include generalized items and generalized rules, respectively. The latter step is relatively straightforward but the former is costly computation.

So far, many approaches have been proposed to quickly discovery all frequent itemsets. ML-T* algorithms [4] were designed to solve a slightly different task of generalized association rule mining. It includes some constraints that each rule contains only items with the same depth, and different minimum supports are set for itemsets with different depths. However, these constraints is not realistic, the rule should come from arbitrary levels of a taxonomy. Three algorithms named *Basic*, *Cumulate* and *Stratification* were introduced in [3]. These algorithms are constructed upon the well-known Apriori algorithm where a traditional horizontal database format is used. Based on taxonomy, *Basic* algorithm first modifies each transaction in a database by adding all generalized items of each item existing in the original transaction. Later, it uses the modified transactions for mining with Apriori-based algorithm. *Cumulate*, an improved variant of *Basic*, filters out meaningless candidates, that are itemsets containing both an item and its ancestor along with taxonomy. *Stratification* uses a sampling method in order to exploit taxonomy information for pruning meaningless candidates. Although *Stratification* is faster than the *Basic* and *Cumulate* algorithms, it is does not completely generate all frequent itemsets. More recently *Prutax* [5] was proposed, to use vertical database format for fastening support counting. It applies right-most depth-first search with hash tree, instead of breath-first search of Apriori-based algorithm, to check for (to avoid generating) meaningless candidates and prunes them out. However, the cost of conditional checking and pruning candidates of this algorithm is expensive.

In this paper, we investigate on this problem and propose a new method to enumerate a set of frequent itemsets without excessive conditional checking. In Section 2, the task of mining generalized association rules is described. Optimization constraints and our set enumeration are explained in Section 3. Out *SET* algorithm is presented in Section 4. In Section 5, a number of experimental results are shown. Finally, a conclusion is made in Section 6.

2. PROBLEM SETTING

The generalized association rule can be formally stated as follows : Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct items, let $T = \{1, 2, \ldots, n\}$ be a set of transaction identifiers (*tids*), and let $D = \{t_j \mid j \in T\}$ be the input database where t_j is the j-th transaction.

A transaction can be represented with a set of items (subset of I), so-called horizontal format. An item can be represented with a set of transactions containing it (subset of T), so-called vertical format.

Let τ be a taxonomy, a directed acyclic graph (tree) on the items. An edge in τ represents is-a relationship. When there is an edge from i_1 to i_2 in τ, i_1 is called a *parent* of i_2 and i_2 is called a child of i_1 (i_1 is a generalization of i_2). An item is called an ancestor of i, denoted $\hat{i}$, when there is a path from $\hat{i}$ to i in τ. Only leaf items of a taxonomy present in database. The other items are called generalized items.

Trans.	Itemsets
1	ADE
2	BE
3	ABDE
4	AC
5	ABCDE
6	C

Items	Tidsets
A	1345
B	235
C	456
D	135
E	1235

Figure 1 : Horizontal (left) and Vertical (right) database formats

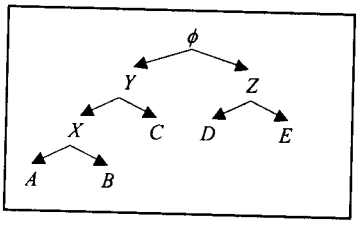

Figure 2 : A taxonomy of items in database

Figure 1 shows an example of a database in two formats. An example of taxonomy of this database is shown in Figure 2. Here, $I = \{A, B, C, D, E, X, Y, Z\}$, and $T = \{1, 2, 3, 4, 5, 6\}$

A set $I_1 \subseteq I$ is called an itemset. The *support* of an itemset I_1, denoted $\sigma(I_1)$, is defined as a number of transactions in which I_1 occurs as a subset. A $t(I_1)$ is defined as a set of transactions which contain I_1 as their subset. An itemset is *frequent* if its support is greater than or equal to a user-specified *minimum support (minsup)* value.

A generalized association rule is an implication of the form R: $I_1 \rightarrow I_2$, where $I_1, I_2 \subseteq I$, $I_1 \cap I_2 = \phi$, and no item in I_2 is an ancestor of any items in I_1. For example, $A \rightarrow C$, $X \rightarrow C$ are generalized association rules, while $A \rightarrow XC$ is not. The *support* of the rule, defined as $\sigma(I_1 \cup I_2)$, is the number of transactions containing both I_1 and I_2. For example, the support of $A \rightarrow C$ is $\sigma(A \cup C) = |t(A) \cap t(C)| = |\{1345\} \cap \{456\}| = |\{45\}| = 2$. The *confidence* of the rule, defined as $\sigma(I_1 \cup I_2)/\sigma(I_1)$, is simply the conditional probability that a transaction contains I_2, given that it contains I_1. For example, the confidence of $A \rightarrow C$ is $\sigma(A \cup C)/\sigma(A) = 2/4$. The rule is *frequent* if its itemset $I_1 \cup I_2$ is frequent. The rule is *confident* if its confidence is greater than or equal to a user-specified *minimum confidence (minconf)* value.

Thus, the task of generalized association rule mining is to discover all rules from arbitrary levels of taxonomy that have support and confidence greater than or equal *minsup* and *minconf* thresholds, respectively. This consists of two main steps. The first step is to find all frequent itemsets and the second step is to generate all high confidence rules. The latter step is relatively straightforward while the former is costly computation. In this work, we will focus on the former step.

3. THE PROPOSED SET ENUMERATION

This section shows three constraints for optimization techniques in calculating all frequent itemsets. Based on these constraints, set enumeration is presented.

3.1 Optimization constraints

The performance of finding frequent itemsets depends on the amount of candidates to be counted, and support counting for each candidate. For real data, the vertical approach, i.e. a method using vertical database format, is shown to outperform the horizontal approaches [2]. In this approach, support counting is simply implemented by intersecting tidsets where the computational cost is less than that in the horizontal approach. The amount of candidates can be reduced by using some constraints in the following lemmas.

Lemma 1. *If an itemset I_1 is infrequent, all supersets of I_1 are infrequent.*

Assume that the support of X is lower than the specified *minsup*. An itemset, which is constructed by joining X with other itemsets, will have support equal to the intersection of the tidset of X and the other one. Therefore, its support is not greater than X, and it is also infrequent.

Lemma 2. *The support of an itemset $\dot{I}_1$ that contains both item i_1 and its ancestor $\hat{i}_1$ is equal to the support of an itemset I_1 which contain only the item i_1.*

In Figure 2, the support of an itemset XAC, equals to the support of an itemset AC. That is, $\sigma(X \cup A \cup C) = |t(X) \cap t(A) \cap t(C)| = t(A) \cap t(C)| = \sigma(A \cup C)$.

Lemma 3. *For an itemset I_1, the support of an itemset $\hat{I}_1$ (an ancestor itemset of I_1) that is generated by replacing one or more items in I_1 with their ancestors, will be greater than or equal to the support of I_1.*

In Figure 2, the support of XC is greater than or equal to the support of AC. That is, $|t(X)| \geq |t(A)|$ and hence $|t(X) \cap t(C)| \geq |t(A) \cap t(C)|$ or $\sigma(X \cup C) \geq \sigma(A \cup C)$.

Lemma 1 shows that it is not necessary to count the supports of superset itemsets of an infrequent itemset. Lemma 2 shows that it is useless to generate itemsets which contain both a certain item and its ancestor. Lemma 3 shows that we need not

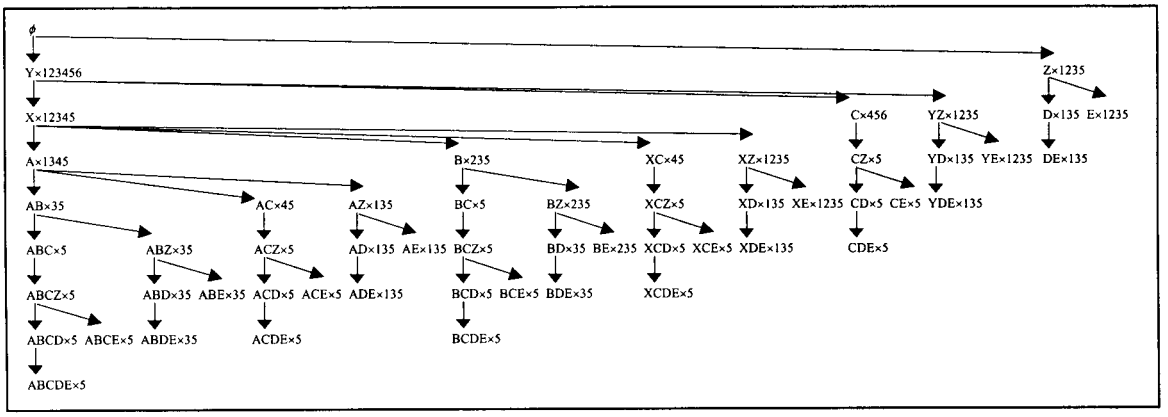

Figure 3 : A complete itemsets tree using a new set enumeration

count the support of any itemset that the ancestor itemsets of which are infrequent. Lemma 1 can be used in both traditional association rule mining and generalized association rule mining while Lemma 2 and Lemma 3 can only be applied to generalized association rule mining.

3.2 Set Enumeration

In the past, previous algorithms including *Basic, Cumulate* and *Prutax* utilize the same set enumeration as used in Apriori [1]. Even they may occupy different orders of set enumeration. The Apriori-based algorithms use breath-first search strategy. *Prutax* uses right-most depth-first search strategy, and then sorts items by increasing generality [5] that is the left items is in the lower level of taxonomy tree than the right one. The former can apply only Lemma 1 and 2 but the latter can invoke all of the constraints (Lemma 1-3) for optimization, However, it uses hash tree for fast checking on frequent itemsets with highly computation. Unlike previous approaches, our new set enumeration implements all three lemmas without intensive checking.

Using the database in Figure 1 and a taxonomy in Figure 2, our set enumeration starts with an empty set. Then, we add all generalized frequent items in the second level of the taxonomy, that are item *Y* and *Z*, and form the second level of an itemsets tree as shown in Figure 3. The children of any itemsets are generated in two ways. First, we generate all *generalized child itemsets*. Each generalized child itemset is generated by replacing the right-most items of those itemsets with one of their children (if exists). Second, we generate all *joined child itemsets* by joining those itemsets with all of their siblings that have higher orders with trivial checking. For example, generating the children of itemset *Y*, we first generate *generalized child itemsets* that is *X* and *C*, and then generate *joined child itemsets*, i.e. *YZ*. In the same way, the *generalized child itemsets* of *X* (i.e. *A* and *B*) and *joined child itemsets* of *X* (i.e. *XC* and *XZ*, replacing *YZ* with *X*) are generated. This process occurs recursively until no itemsets are generated. Finally, a complete itemsets tree is constructed without excessive checking cost shown in Figure 3.

4. ALGORITHM

We propose a new algorithm, called *SET*, for finding frequent itemsets tree based on the set enumeration shown in Section 3.2. Figure 4 shows pseudo-code of *SET*.

```
SET-MAIN (Datbase,Taxonomy,minsup):
1. Root = Null Tree // Root node of set enumeration
2. Addlink(Root, All frequent items from second level of taxonomy)
3. SET-EXTEND(Root)

SET-EXTEND(Father):
4. For i = 1 to numlinks(Father)
5.    GTree=Null Tree
6.    For each child of (Last(Fi) //Generate generalized child itemset
7.       C = Replace(Last(Fi), Childj(Last(Fi)))
8.       If supp(C) ≥ minsup then Addlink(GTree, C)
9.    For j = i+1 to numlinks(Father) //Generate joined child itemset
10.      C = Fi ∪ Fj
11.      If supp(C) ≥ minsup then Addlink(GTree, C)
12.   Father.Links[i]Child = Gtree
13.   If GTree!=NULL then SET-EXTEND(Father.Links[i].Child)
```

Figure 4 : The *SET* Algorithm

In this Figure, the main procedure is *SET-MAIN* and a function, called *SET-EXTEND*, creates a subtree follow by a proposed set enumeration. *SET-EXTEND* is executed recursively to create all descendant nodes under the root node. The *Addlink* function creates a child node of a parent node, such as *Addlink*(*Y*,*X*) creates a child node *X* of a parent node *Y*. The *Last* function in line 6 returns the last item of an itemset. For example, *Last*(*XY*) returns *Y*. The if statement in line 8 and 11 prunes nodes with supports less than *minsup* (i.e. infrequent).

5. EXPERIMENTAL RESULTS

The proposed algorithm is evaluated and compared with *Prutax*, which is the best algorithm at the present time [5]. All experiments were made on a 700 MHz Duron PC with 256 MB of memory. The OS environment is Microsoft Windows 98. *Prutax* and *SET* algorithms are coded in C++ language.

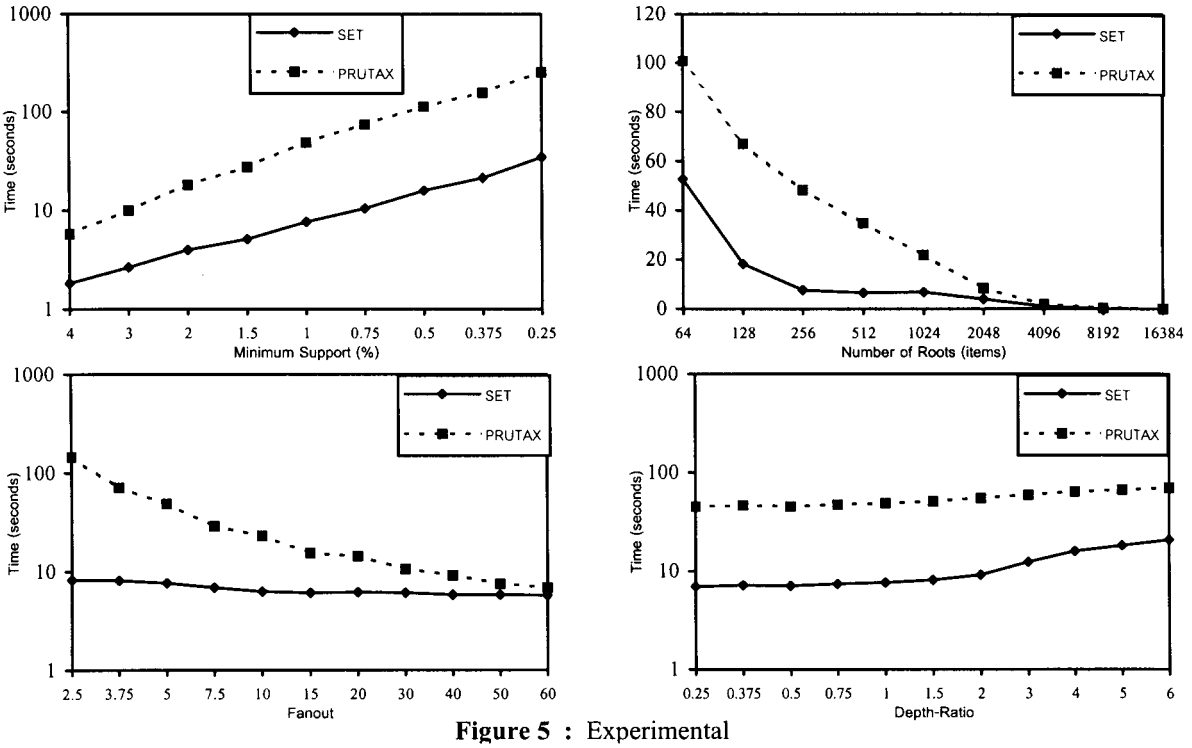

Figure 5 : Experimental

Synthetic datasets are used in our experiments. They were automatically generated by the generator tool provided at IBM Almaden site but slightly modified default values as shown in [5]. Details of each parameter can be found in [3]. The important default values of parameters in the datasets are shown in Table 1.

Parameter	Default
Number of transactions	100K
Average size of the transaction	10
Number of items	100K
Number of roots	250
Fanout	5
Depth-ratio	1
Minimum support	1%

Depth-ratio $\approx \left(\frac{\text{probability that item in a rule comes from level i}}{\text{probability that item comes from level i+1}}\right)$ [3]

Table 1 : The default values of parameters in the datasets

Four experiments were made to investigate the performance of *SET* compared with *Prutax* algorithm. We varied four parameters : minimum support, number of roots, fanout and depth-ratio. The experimental results are shown in Figure 5. With different *minsup*, *SET* runs 2-8 times faster than *Prutax*. *SET* relatively performs well when the datasets have less number of roots, that is the actual situation in real-life database. *SET* can reduce more cost in the case of lower fanout, i.e. deep taxonomy. *SET* achieves approximately 4-6 times better than *Prutax* with depth-ratio variation.

6. CONCLUSION

This paper gives a result of investigating optimization constraints using in various algorithms for finding all frequent itemsets of generalized association rule mining. We propose a new set enumeration for generating frequent itemsets without intensive checking. The performance of our approach is shown via a set of experiments by varying minimum support, number of roots, fanout and depth-ratio As the result, our approach is better than *Prutax* in all cases.

ACKNOWLEDGEMENTS

This paper has been supported by Thailand Research Fund (TRF) and National and Computer Technology Center (NECTEC) under project number NT-B-06-4F-13-311 (on 2001).

REFERENCES

[1] R.Agrawal, T. Imielinski, A. Swami. "Mining Association Rules between Sets of Items in Large Databases," In Proc. of ACM SIGMOD '93, 1993, Washington, USA.

[2] M. J. Zaki and C.-J, Hsiao, "CHARM: An efficient algorithm for closed association rule mining," Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Institute, October 1999.

[3] R. Srikant, R. Agrawal, "Mining Generalized Association Rules," In Proc. of the VLDB '95, 1995, Zürich, Switzerland.

[4] J. Han, Y. Fu, "Discovery of Multiple-Level Association Rules," In Proc. of the VLDB '95, 1995, Zürich, Switzerland.

[5] J. Hipp, A. Myka, R. Wirth, and U. Güntzer. "A new algorithm for faster mining of generalized association rules," In Proc. 2nd PKKD, 1998.