A New Method for Finding Generalized Frequent Itemsets in Generalized Association Rule Mining

Kritsada Sriphaew, Thanaruk Theeramunkong

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: kong@siit.tu.ac.th, ping@siit.tu.ac.th

Abstract

Generalized association rule mining is an extension of traditional association rule mining to discover more informative rules, given a taxonomy. In this paper, we describe a formal framework for the problem of mining generalized association rules. In the framework, The subset-superset and the parent-child relationships among generalized itemsets are introduced to present the different views of generalized itemsets, i.e. the lattice of generalized itemsets and the taxonomies of k-generalized itemsets ,respectively. We present an optimization technique to reduce the time consuming by applying two constraints each of which corresponds to each view of generalized itemsets. In the mining process, a new set enumeration algorithm, named SET, that utilizes these constraints to fasten mining all generalized frequent itemsets is proposed. By experiments on synthetic data, the results show that SET outperforms the current most efficient algorithm, Prutax, by an order of magnitude or more.

1. Introduction

In the area of Knowledge Discovery in Databases (KDD), association rule mining is one of the important tasks. It was first introduced in [1] to find the set of all subsets of items (called itemsets) that frequently occur in many database records or transactions, and to extract the rules telling us how a subset of items influences the presence of another subset [2]. Nevertheless, association rules may not provide desired knowledge in the database. It may be limited with the granularity factors over the items. For example, suppose that the database keeps a set of transactions, where chocolate milk tends to be purchased together with wheat bread. We may obtain a rule of "5% of customers who buy wheat breads, also buy chocolate milk". At this point, it is more intuitive or more informative to have a rule like "30% of customers who buy bread, also buy milk" instead of the previous one.

For this purpose, generalized association rule mining (GARM) was developed [3]. In GARM, a taxonomy (is-a hierarchy) over the items is available. In the supermarket scenario, the taxonomy can classify products (or items) into brands, categories, product groups, and so forth. Only leaf-level items of a taxonomy are presented in the database. Together with a taxonomy, the database can be used to mine more informative, initiative and flexible rules (called generalized association rules) than the traditional association rules.

The problem of mining generalized association rules was first introduced in [3]. In this work, five algorithms named Basic, Cumulate, Stratify, Estimate and EstMerge were proposed to solve this problem. All of these algorithms use the horizontal database format and the breath-first search manner as in Apriori-based algorithm [2]. Basic first extends each transaction in a database by adding all distinct generalized items of each items existing in the original transaction, and then generates itemsets without pruning meaningless itemsets (itemsets containing both an item and its ancestor according to taxonomy). Cumulate, an improved variant of Basic, incrementally filters out the meaningless itemsets before counting their supports. Stratify exploits more taxonomy information. This method can reduce the number of generalized itemsets to be counted. Nevertheless, Stratify wastes a lot of time in scanning the database multiple times. Later, Estimate was proposed to get rid of this waste by using a sampling method to estimate the support of generalized itemsets. However, the extra pass to count the generalized itemsets, that may wrongly be expected

not to satisfy the minimum support, has executed. EstMerge is an improvement of Estimate by postponing this extra pass in the step of calculating k-generalized frequent itemsets to the next step of calculating (k+1)-generalized frequent itemsets. This action results in reducing the number of scanning database.

More recently efficient algorithm named Prutax was proposed in [4]. Instead of horizontal database format, the vertical database format is applied to reduce the time needed for scanning database multiple times. Prutax aplies the right-most depth-first search manner. Instead of "*generate and test*" approach as done in the previous algorithms, Prutax avoids generating meaningless itemsets by using hash tree checking. In this approach, the generalized itemsets containing the items in the higher-level of taxonomy are always evaluated before its descendants, which reduces the number of generalized itemsets to be calculated. However, the limitation of Prutax is the cost of checking. Each generalized itemset, which is generated, have to be checked whether its ancestor itemsets are frequent or not by using hash tree before counting its support.

The main limitation of almost all proposed algorithms [3,4] is that they make multiple passes over the disk-resident database incurring high I/O overheads. Moreover, these algorithms omit some useful information of taxonomy for optimization, and the cost of checking in Prutax algorithm adds more time consuming instead of reducing it. Our work aims to overcome these limitations.

The rest of this paper is organized as follows. The problem of GARM is formally described in Section 2. In Section 3, two different views: the lattice of generalized itemsets and the taxonomies of k-generalized itemsets are presented. Two optimization constraints and our set enumeration, *SET* algorithm, are proposed in Section 5. In Section 6, the performance of *SET* is evaluated on synthetic datasets with some variations. The paper ends with a conclusion in Section 7.

2. Problem Statement

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 an input database where t_j is the j-th transaction. A transaction can be represented with a set of items (a subset of I), so-called horizontal format while an item can be represented with a set of transactions containing it (a subset of T), so-called vertical format as shown in figure 1.

Let τ be a taxonomy, a directed acyclic graph (tree) on the items. An edge in τ represents is-a (parent-child) 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. An item is called an *ancestor* of i, denoted $\hat{i}$, when there is a path

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

Items	Tidsets
A	*1345*
B	*235*
C	*3456*
D	*15*
E	*1235*

Figure 1. Horizontal (left) and vertical (right) database

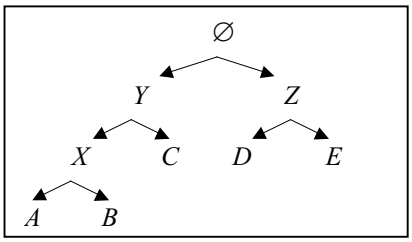

Figure 2. A taxonomy on items in database

from $\hat{i}$ to i in τ. In conversely, i is called a descendant of $\hat{i}$. Only leaf items of a taxonomy are presented in the database. An example of taxonomy is shown in figure 2.

A set $I_G \subseteq I$ is called a *generalized itemset* when I_G does not contain both item and its ancestor. A $t(I_G)$ is defined as a set of transactions which contain I_G as their subset. The tids of parent items are given by the union in tids of its child items. From figure 1 and 2, *AB*, *AZ*, *AD* are generalized itemsets while AX, XY are not, and $t(X) = t(A) \cup t(B)$. The *support* of I_G, denoted $\sigma(I_G)$, is defined as a percentage of transactions in which I_G occurs as a subset to the total transactions, thus $\sigma(I_G) = |t(I_G)|/|T|$. A generalized itemset is called *generalized frequent itemset* if its support is greater than or equal to a user-specified *minimum support (minsup)* threshold.

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, consider a database in figure 1 and a taxonomy in figure 2, $A \rightarrow C$ and $X \rightarrow C$ 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 percentage of transactions containing both I_1 and I_2 to the total transaction. For example, the support of $A \rightarrow C$ is $\sigma(A \cup C) = |t(A) \cap t(C)| / |T| = |\{1345\} \cap \{456\}|/6 = |\{45\}|/6 = 2/6$ or 33%. 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 $\sigma(A \cup C)/\sigma(A) = 2/4$ or 50%. The rule is called *generalized association rule* if its confidence is greater than or equal to a user-specified *minimum confidence (minconf)* threshold.

The task of GARM is to discover all rules from arbitrary levels of taxonomy that have support and confidence greater than or equal to *minsup* and *minconf*

thresholds, respectively. This consists of two main steps : 1) find all frequent itemsets, and 2) generate all high confidence rules. The latter step is relatively straightforward while the former is costly computation and I/O intensive. Thus, the problem of GARM can be reduced to the problem of finding frequent itemsets. In this work, we will focus on this problem.

3. Generalized itemset relationships

The generalized itemsets contain both subset-superset relationship and parent-child relationship according to taxonomy that can be represented by the lattice of generalized itemsets and the taxonomies of k-generalized itemsets, respectively.

3.1. Subset-superset relationship: lattice of generalized itemsets

Due to the space limitation, we assume that the reader is familiar with basic concepts of lattice theory. However, more details can be found in [6]. The formal concept analysis [7] and the formal concept of itemset lattice in association rule mining [8, 9, 10, 11] can be adapted to construct the generalized itemset lattice in GARM. Therefore, the following formal definitions are useful for describing the concept of generalized itemset lattice.

Definition 1 (Set union and intersection according to taxonomy operation): The *set union according to taxonomy operation*, denoted by $\cup_T$ is a binary operation, which is produced by the set union and contains only the most descendant items according to taxonomy. For example, $AC \cup_T BC = ABC$, $AD \cup_T AW = AD$, and $UD \cup_T AW = AD$. The *set intersection according to taxonomy operation*, denoted by $\cap_T$ is a binary operation, which is produced by the set intersection and contains only the most descendant items according to taxonomy. For example, $AC \cap_T BC = C$, $AD \cap_T AW = AD$, and $UD \cap_T AW = AD$.

Definition 2 (Lattice of generalized itemsets): The lattice of generalized itemsets is the partial order specified by the subset relation $\subseteq$, where the meet is given by the set intersection operation, and the join is given by the set union according to taxonomy operation as follows. For any $X_1, X_2 \subseteq I$,

$$Meet: \ X_1 \wedge X_2 = (X_1 \cap_T X_2)$$

$$Join: \ X_1 \vee X_2 = (X_1 \cup_T X_2)$$

3.2. Parent-Child Relationship: taxonomies of k-generalized itemsets

The useful definitions for defining the taxonomies of k-generalized itemsets are described as follows:

Definition 3 (Ancestor-Descendant Itemset): Let $\hat{X}$, $X \subseteq$ I, The itemset $\hat{X}$ is an *ancestor itemset* of X if $|\hat{X}| = |X|$ and $\hat{X}$ can be generated by replacing one or more items in X with one of their ancestors items. X is called a *descendant itemset* of $\hat{X}$.

Definition 4 (Parent-Child Itemset): Let $\hat{X}$, $X, X' \subseteq$ I, $\hat{X}$ is a *parent itemset* of X if there is no X' with X' being an ancestor itemset of X and $\hat{X}$ being an ancestor itemset of $\hat{X}$. X is called a *child itemset* of $\hat{X}$.

Normally, the given taxonomy presents only the relation of single items (not itemsets) in the database. Let's call 1-itemset taxonomy. The relation of these items is *parent-child relationship* according to taxonomy as shown by the connection lines between parent and child itemsets. By using parent-child relationship, we can extend the original taxonomy to express the k-generalized itemsets.

Each k-generalized itemset has the parent-child itemset relationship. The tids of parent itemset equals to the union of tidsets of their child itemsets. For example, the itemset *ACZ* in 3-generalized itemset taxonomy has two child itemsets, i.e. *ACD* and *ACE*, by which $t(ACZ) = t(ACD) \cup t(ACE) = \{5\} \cup \{35\} = \{35\}$.

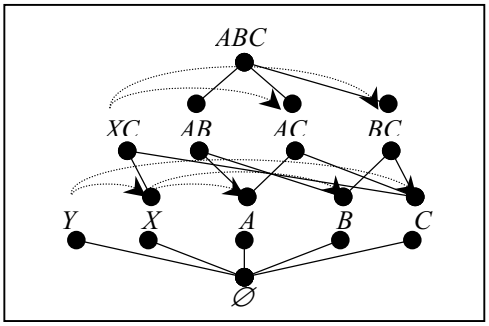

Figure 3. Combination of two relationships (A part)

3.3. Combination of two relationships

The generalized itemsets can be shown in the complex view that combines both subset-superset relationship and parent-child relationship. For example, if we consider only the generalized items *A*, *B*, *C*, *X*, *Y* and the taxonomy as in figure 2. The complex view of generalized itemset lattice is shown in figure 3. The thick lines show the subset-superset relationship, and the dash arrow lines show the parent-child relationship by which the itemset located at the beginning of an arrow is the parent itemset of the itemset located at the end of the arrow.

4. Discovery of generalized frequent itemsets

Most of computational cost is to count the support of generalized itemsets for checking whether they are frequent or not, and checking for non-generating meaningless itemsets. To reduce this computational cost, the constraints and techniques are applied to optimize the number of generalized itemsets to be counted.

4.1. Constraints on generalized itemsets

Two lemmas are presented to justify the optimization.

Lemma 1. For any $X \subseteq I$, if a generalized itemset X is frequent, all subsets of X are frequent. Dually, if a generalized itemset X is infrequent, all supersets of X are infrequent.

Proof: Let $X,Y, Z \subseteq I$ and $Z = XY$. The support of Z, $\sigma(Z) = |t(Z)|=|t(X)\cap t(Y)|$ must be less than or equal to the supports of its subsets, i.e. X and Y. Thus, if Z is frequent, X and Y are too. If both X and Y or either of them is infrequent, then neither does Z.

Lemma 2. For any $X, \hat{X} \subseteq I$ where $\hat{X}$ is an ancestor itemset of X, if $\hat{X}$ is frequent, then X is also frequent. Dually, if X is infrequent, $\hat{X}$ is also infrequent.

Proof: Let $x, \hat{x} \in I$ and $Y, Z, \hat{Z} \subseteq I$. Assume that $Z = xY$, $\hat{Z} = \hat{x}\,Y$. $\hat{x}$ is an ancestor item of x, and $\hat{Z}$ is an ancestor itemset of Z. The support of $\hat{Z}$,$\sigma(\hat{Z})=|t(\hat{Z})|=|t(\hat{x})\cap t(Y)|$, must be greater than or equal to the support of Z, $\sigma(Z) = |t(Z)| = |t(x)\cap t(Y)|$, since the support of ancestor item $\hat{x}$ is greater than or equal to the support of x. Thus, If $\hat{Z}$ satisfies *minsup* (frequent), Z does also. If Z does not satisfy *minsup* (infrequent), $\hat{Z}$ does also.

For fast finding all generalized frequent itemsets, each lemma can be applied to each relationship of generalized itemsets. Lemma 1 concerns with the subset-superset relation which exists in the lattice of generalized itemset. Lemma 2 concerns with the ancestor itemset which exits in the parent-child relation in the taxonomies of k-generalized itemsets. These two lemmas can be used to avoid generating infrequent itemsets. For efficient traversal, we try to generate only generalized frequent itemsets. From Lemma 1, all subsets of any generalized itemsets must be ensured that they are frequent before generating them. From Lemma 2, the ancestor itemsets must be frequent before generating their child itemsets.

4.2. *SET* algorithm

In this section, a new set enumeration named *SET* is proposed for finding all generalized frequent itemsets. With vertical database format, our method enumerates all generalized frequent itemsets using left-most depth first search. *SET* algorithm applies two efficient approaches that is: 1) Based on combination of two relationships, describing in section 3.3, we construct our novel set enumeration that can avoid intensive checking on meaningless itemsets. 2) Two constraints are implemented to prevent counting infreqeunt generalized itemsets.

Using the vertical database in figure 1 and a taxonomy in figure 2 with *minsup*=1/6, our set enumeration starts with an empty set. Then, we add all generalized frequent items in the second level of the taxonomy, that are *Y* and *Z*, and form the second level of tree as shown in figure 4. The children of any itemsets are generated in two manners. First, we generate all *tax-based child itemsets* (based on parent-child relationship). Each generalized

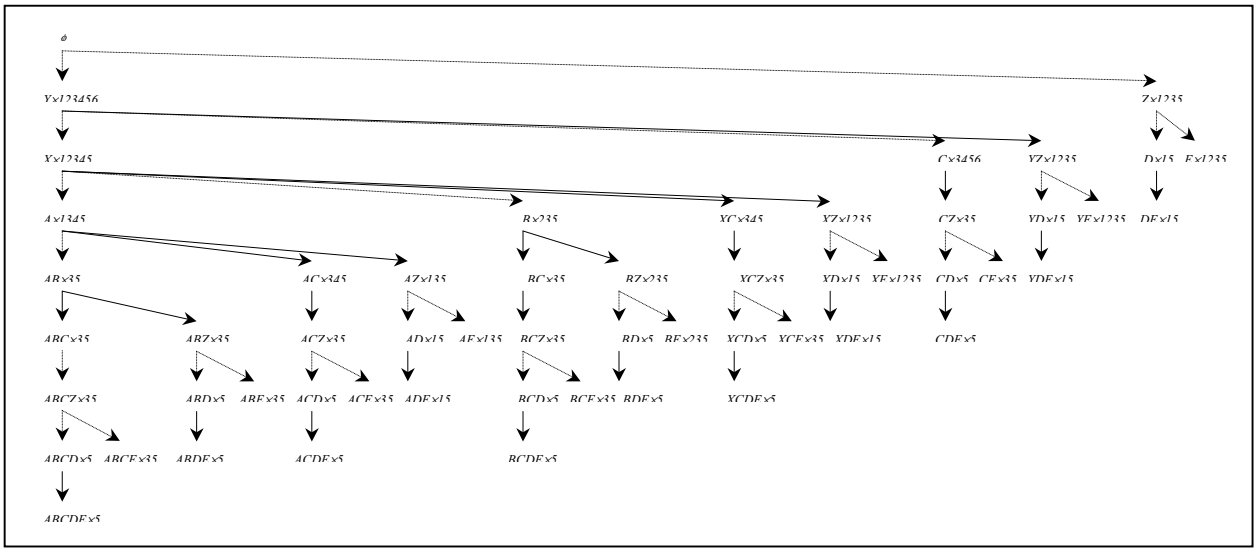

Figure 4. A complete itemsets tree using a new set enumeration

child itemset is generated by replacing the right-most items of those itemsets with one of their children (if exists). Second, we generate all *join-based child itemsets* (based on subset-superset relationship) by joining those itemsets with all of their siblings that have higher orders. For example consider on itemset *Y*, we first generate tax-based child itemsets that is *X* and *C*, and join-based child itemsets, i.e. *YZ*. In the same way, the tax-based child itemsets of *X* (i.e. *A* and *B*) and join-based child itemsets of *X* (i.e. *XC* and *XZ*, replacing *YZ* with *X*) are generated. This process recursively occur until no new generalized itemsets are generated. Finally, a complete itemset tree is constructed without excessive checking cost as in figure 4.

The formal pseudo-code of *SET* are shown in figure 5. 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 itemsets under the root itemsets. The *Addlink* function creates a child itemset of a parent itemset, such as *Addlink*(*Y*,*X*) creates a child itemset *X* of a parent itemset *Y*. The *Last* function in line 6 returns the last item of a generalized itemset. For example, *Last*(*XY*) returns *Y*. The Tax-Child function in line 7 returns the tax-based child itemsets of F_i. For example, to generate the tax-based child itemset of *XZ*, when *D* and *E* are child items of *Z*, the functions Tax-Child(*XZ*,*D*) and Tax-Child(*XZ*,*E*) produce *XD* and *XE*, respectively. For the if statements in line 8 and 11 prune nodes with supports less than *minsup* (i.e. infrequent).

```
SET-MAIN (Database, 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 tax-based child itemset
7.         C = Tax-Child(Fi, Child(Last(Fi)))
8.         If supp(C) ≥ minsup then Addlink(GTree, C)
9.    For j = i+1 to numlinks(Father) //Generate join-based 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 5. The pseudo-code of *SET* algorithm

5. Experimental Results

The *SET* algorithm is evaluated and compared with Prutax. All algorithms are coded in C language and the experiments were made on a 1 GHz Pentium III with 1 GHz of main memory running Windows 2000.

As our preliminary experiments, the synthetic datasets are used. The synthetic datasets were automatically generated by the generator tool provided at IBM Almaden with slightly modified default values. The important default parameters in the datasets are shown in Table 1.

Table 1. Default values of parameters in the datasets

Parameter	Default
Number of transactions	1000K
Average size of the transaction	10
Number of items	100K
Number of roots	250
Fanout	5
Depth-ratio	1
Minimum support	1%

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

Six experiments were made to investigate the performance of *SET* algorithm compared with Prutax algorithm by changing a different parameter in each experiment. All parameters except the one being varied were set to their default values. Four parameters, i.e. minimum support, number of roots, fanout and depth-ratio, are varied to investigate the algorithm. We also scale-up the datasets by varying two parameters, i.e. number of transactions and number of items. The experimental results are shown in figure 6.

From figure 6, *SET* runs faster than Prutax with different *minsup*. The number of generalized frequent itemsets increase when lower minimum support. This cause effects to Prutax by increasing depths of hash tree and more time consuming for checking is needed, while *SET* needs not to checking. In case of less number of roots, the increasing of taxonomy levels effects to the large number of ancestor itemsets. *SET* doesn't effect to this situation while Prutax requires more computational time for checking. With different fanouts, the child of each item in taxonomy are varied. The number of ancestor itemsets in lower fanouts is larger than higher fanouts which makes *SET* performs better than Prutax. SET achieves approximately 4-6 times better than Prutax with depth-ratio variation. In the lower depth ratio, more rules have items come from the lower parts rather than the upper parts of taxonomy, such that the ancestor itemsets are increasing to gather the time consuming for checking in Prutax.

In scale-up, *SET* performs well with large number of transactions, since the generalized frequent itemsets increase which make more intensive checking in Prutax. At last, we increase the number of items (including ancestor items) from 10,000 to 1,000,000 items. *SET* does not effect to this variation, since the items are sparseness in transactions with larger number of items. Thus, the number of generalized frequent itemsets is reduced.

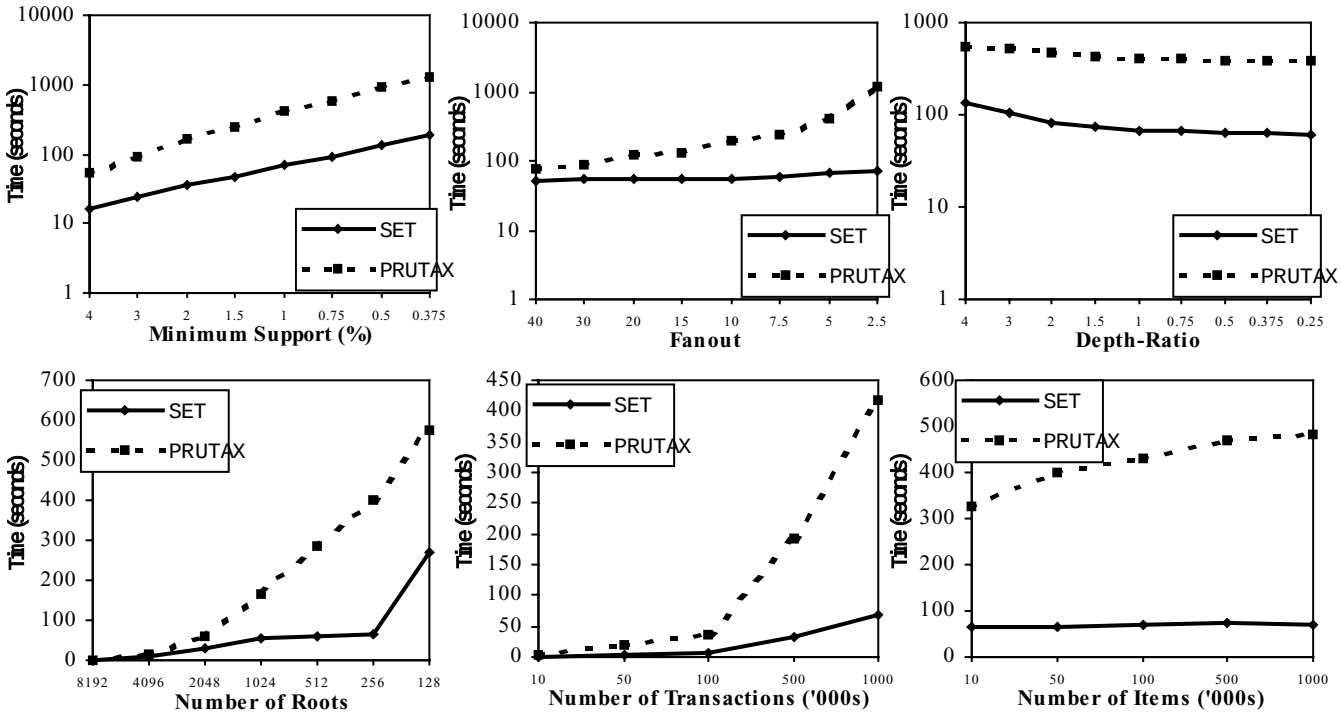

Figure 6. Experimental results

6. Conclusion

In this paper we presented a theoretical framework of generalized itemsets based on two relationships: (1) subset-superset relationship (represented by lattice of generalized itemsets), and (2) parent-child relationship (represented by taxonomy of k-generalized itemsets). To efficiently discover all generalized frequent itemsets, we applied two constraints for these two relationships. We proposed a *SET* algorithm to enumerate all generalized frequent itemsets. *SET* uses the novel traversal on the combination of two relationships to avoid generating meaningless itemsets, and applies two constraints to prevent counting useless generalized itemsets that are infrequent. The investigation on experiments shows that our proposed set enumeration, *SET* algorithm, can reduce the cost of intensive checking as in the current most efficient algorithm, Prutax algorithm, and prevents overhead in enumerating generalized frequent itemsets. From these causes, *SET* algorithm can fasten finding all generalized frequent itemsets in generalized association rule mining task.

7. Acknowledgment

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

8. References

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

[2] R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules", *VLDB '94*, Santiago Chile, 1994.

[3] R. Srikant, and R. Agrawal, "Mining Generalized Association Rules", *VLDB '95*, Zürich Switzerland, 1995.

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

[5] C.L. Lui, and F.L. Chung, "Discovery of Generalized Association Rules with Multiple Minimum Supports", *4th PKDD*, Lyon France, Sept. 2000, pp.510-515.

[6] B.A. Davey, and H.A. Priestly, "Introduction to Lattices and Order", Cambridge University Press, 1990.

[7] B. Ganter, and R. Wille, "Formal Concept Analysis: Mathematical Foundations", Springer-Verlag, 1999.

[8] M.J. Zaki, S. Parthasaarathy, M. Ogihara, and W. Li, "New Algorithms for Fast Discovery of Association Rules", *KDD'97*, Newport Beach California, 1997.

[9] M.J. Zaki, and M. Ogihara, "Theoretical foundatations of association rules", *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, June 1998.

[10] 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.

[11] M.J. Zaki, "Scalable algorithms for association mining", *IEEE Trans Knowledge & Data Engineering*, 12(3):372-390, 2000.