PAPER

Fast Algorithms for Mining Generalized Frequent Patterns of Generalized Association Rules

Kritsada SRIPHAEW[†], *Student Member and* **Thanaruk THEERAMUNKONG**[†], *Member*

SUMMARY Mining generalized frequent patterns of generalized association rules is an important process in knowledge discovery system. In this paper, we propose a new approach for efficiently mining all frequent patterns using a novel set enumeration algorithm with two types of constraints on two generalized itemset relationships, called subset-superset and ancestor-descendant constraints. We also show a method to mine a smaller set of generalized closed frequent itemsets instead of mining a large set of conventional generalized frequent itemsets. To this end, we develop two algorithms called *SET* and *cSET* for mining generalized frequent itemsets and generalized closed frequent itemsets, respectively. By a number of experiments, the proposed algorithms outperform the previous well-known algorithms in both computational time and memory utilization. Furthermore, the experiments with real datasets indicate that mining generalized closed frequent itemsets gains more merit on computational costs since the number of generalized closed frequent itemsets is much more smaller than the number of generalized frequent itemsets.

key words: *data mining, knowledge discovery, generalized association rule, Galois lattice*

1. Introduction

Association rule mining (ARM) is one of the important tasks in the area of Knowledge Discovery in Databases (KDD). ARM ([1], [7], [20]) is a process to find the set of all subsets of items (called itemsets) that frequently occur in the database records or transactions, and then to extract the rules telling us how a subset of items influences the presence of another subset. However, association rules may not provide desired knowledge in the database. It may be limited with the granularity over the items. For example, a rule "5% of customers who buy wheat breads, also buy chocolate milk" is less expressive and less useful than a more general rule "30% of customers who buy bread, also buy milk". For this purpose, generalized association rule mining (GARM) was developed using the information of a pre-defined taxonomy over the items. The taxonomy is a piece of knowledge, e.g. the classification of the products (or items) into brands, categories, product groups, and so forth. Given a taxonomy where only leaf nodes (leaf items) present in the transactional database, more informative, initiative and flexible rules (called generalized association rules) can be mined from the database. Each generalized association rule contains items from any level of a taxonomy. Similar to ARM, the most important problem of GARM is how to efficiently find all generalized frequent itemsets, which is the computational intensive step.

In the past, there were still few works related to GARM. Most of them focus to fasten mining generalized frequent itemsets. In [14], five algorithms named Basic, Cumulate, Stratify, Estimate and EstMerge were proposed. These algorithms apply the horizontal database and breath-first search strategy like Apriori-based algorithm ([2]). They use the extended database, constructed by adding all distinct ancestors of each items existing in its original transaction, to mine all generalized frequent itemsets. Most methods in GARM exploit some constraints among itemsets for pruning, and discarding meaningless itemsets, i.e. the itemsets containing both an item and its ancestor according to the given taxonomy. However, these algorithms waste a lot of time in multiple scanning the database even the sampling method is applied. As a more recently efficient algorithm, Prutax ([8]) applies a so-called vertical database format to reduce the computational time needed for multiple scanning the database. Instead of "generate and test" as done in previous algorithms, it avoids to generate meaningless itemsets by using hash tree checking. Nevertheless in Prutax, the limitation is the cost of checking whether their ancestor itemsets are frequent or not by using hash tree before counting its actual support. By the way, there exists a slightly different task for dealing with multiple different minimum support in different levels of itemsets as shown in [6] and [10]. A parallel algorithm has also been proposed in [13]. Some recent applications that utilize a GARM are shown in [11] and [9]. Our preliminary research related to GARM is shown in [15].

In this work, we propose a new approach for efficient finding all generalized frequent itemsets using two types of constraints on two generalized itemset relationships, called *subset-superset* and *ancestor-descendant* constraints. We show that it is sufficient to mine only a small set of generalized closed frequent itemsets instead of mining a large set of conventional generalized frequent itemsets. Two algorithms, named *SET* and *cSET*, are proposed to efficiently find generalized frequent itemsets and generalized closed frequent itemsets, respectively.

2. Generalized Association Rules and Generalized Frequent Itemsets

With the presence of a taxonomy, the formal problem description of generalized association rule is different from earlier works on association rule mining ([1], [2]). For clarity, all explanations in the section are illustrated using an

Manuscript received January 23, 2003.
Manuscript revised June 29, 2003.

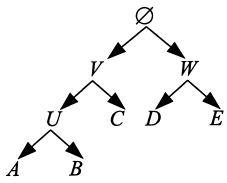
[†]The authors are with Sirindhorn International Institute of Technology, Thammasat University, Bangkadi, Muang, Pathumthani, 12000 Thailand.

Trans	Itemsets
1	ACDE
2	ABC
3	BCDE
4	ACD
5	ABCDE
6	BCDE

(A): Horizontal Database

Items	Tidsets
A	1245
B	2356
C	123456
D	13456
E	1356

(B): Vertical Database



(C): Taxonomy

Items	Tidsets
A	1245
B	2356
C	123456
D	13456
E	1356
U	123456
V	123456
W	13456

(D): Extension Vertical Database

Fig. 1 An example of databases and taxonomy.

example shown in Fig. 1.

Let $\mathcal{T}$ be a taxonomy, a directed acyclic graph on items, which represents *is-a* relationship by edges (e.g. Fig. 1 (C)). The items in $\mathcal{T}$ are composed of a set of leaf items (I_L) and a set of non-leaf items (I_{NL}). Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of distinct items where $I = I_L \cup I_{NL}$, and let $T = \{1, 2, \ldots, n\}$ be a set of transaction identifiers (*tids*). In this example, $I_L = \{A, B, C, D, E\}$, $I_{NL} = \{U, V, W\}$, $I = I_L \cup I_{NL} = \{A, B, C, D, E, U, V, W\}$, and $T = \{1, 2, 3, 4, 5, 6\}$. A subset of I is called an *itemset* and a subset of T is called a *tidset*. Normally, a transactional database is represented in the horizontal database format, where each transaction corresponds to an itemset (e.g. Fig. 1 (A)). An alternative to the horizontal database format is the vertical database format, where each item corresponds to a tidset which contains that item (e.g. Fig. 1 (B)). Note that the original database contains only leaf items. It is possible to represent an original vertical database by extending it to cover non-leaf items where a transaction of which item also supports its related items from taxonomy (e.g. Fig. 1 (D)). Let the binary relation $\delta \subseteq I \times T$ be an extended database. For any $x \in I$ and $y \in T$, $x\delta y$ can be denoted when x is related to y in the database (called x is supported by y). Here, except the elements in I, lower case letters are used to denote items and upper case letters for itemsets.

For $\hat{x}, x \in I$, $\hat{x}$ is an *ancestor* of x (conversely x is a *descendant* of $\hat{x}$) when there is a path from $\hat{x}$ to x in $\mathcal{T}$. For any $x \in I$, a set of all its ancestors (descendants) is denoted by $\mathcal{ANC}(x)$ ($\mathcal{DES}(x)$). For example, $\mathcal{ANC}(B) = \{U, V\}$ and $\mathcal{DES}(W) = \{D, E\}$.

A generalized itemset G is an itemset each element of which is not an ancestor of the others, $G = \{i \in I | \forall j \in G, i \notin \mathcal{ANC}(j)\}$. For example, $\{A, B\}$ (AB for short), $\{A, W\}$ (AW for short) are generalized itemsets. Let $I_G = \{G_1, G_2, \ldots, G_l\}$ be a finite set of all generalized itemsets. Note that, for $1 \leq i \leq l$, $G_i \subseteq I$ and $I_G \subseteq \mathcal{P}(I)$. The support of G, denoted by $\sigma(G)$, is defined by a percentage of the number of transactions in which G occurs as a subset to the total number of transactions, thus $\sigma(G) = |t(G)|/|T|$. Any G is called *generalized frequent itemset* (GFI) when its support is at least a user-specified *minimum support* (*minsup*) threshold.

In GARM, a *meaningful* rule is an implication of the form $\mathcal{R}$: $G_1 \rightarrow G_2$, where $G_1, G_2 \in I_G$, $G_1 \cap G_2 = \emptyset$, and no item in G_2 is an ancestor of any items in G_1. For example, $A \rightarrow C$ and $U \rightarrow C$ are meaningful rules, while $A \rightarrow UC$ is a *meaningless* rule because its support is redundant with $A \rightarrow C$. The *support* of a rule $G_1 \rightarrow G_2$, defined as $\sigma(G_1 \cup G_2) = |t(G_1 \cup G_2)|/|T| = |t(G_1) \cap t(G_2)|/|T|$, is the percentage of the number of transactions containing both G_1 and G_2 to the total number of transactions. The *confidence of a rule*, defined as $\sigma(G_1 \cup G_2)/\sigma(G_1)$, is the conditional probability that a transaction contains G_2, given that it contains G_1. For example, the support of $A \rightarrow C$ is $\sigma(A \cup C) = |t(A) \cap t(C)|/|T| = |1245|/6 = 4/6$ or 67% and the confidence is $\sigma(A \cup C)/\sigma(A) = 1$ or 100%. The meaningful rule is called a *generalized association rule* (GAR) when its confidence is at least a user-specified *minimum confidence* (*minconf*) threshold.

The task of GARM is to discover all GARs the supports and confidences of which are at least minsup and minconf, respectively.

3. Two Relationships on Generalized Itemsets

This section introduces two relationships, i.e. subset-superset and ancestor-descendant relationships, based on lattice theory. For more details about lattice theory, the reader can refer to [4]. To construct the generalized itemset lattice in GARM, we adapt the formal concept analysis ([5]) and itemset lattice in ARM ([19]). Similar to ARM, GARM occupies the subset-superset relationship which represents a lattice of generalized itemsets. As the second relationship, an ancestor-descendant relationship is originally introduced in this work to represent a set of k-generalized itemset taxonomies.

3.1 Subset-Superset Relationship: Lattice of Generalized Itemsets

Definiton 1 (Subset-superset relationship): Let a binary relation $\delta_{\mathcal{S}} \subseteq \mathcal{P}(I) \times \mathcal{P}(I)$ be the *subset-superset relationship*. For any $X_1, X_2 \in I_G$, $X_1 \delta_{\mathcal{S}} X_2$ is denoted when X_1 is a subset of X_2 (X_2 is a superset of X_1).

Definiton 2 (Lattice of generalized itemsets): The lattice of generalized itemsets is the partial order specified by a subset-superset relationship $\delta_{\mathcal{S}}$, where the meet is given by the set intersection on generalized itemsets, and the join is given by the set union on generalized itemsets as follows. For any $X_1, X_2 \in I_G$,

$$Meet: X_1 \wedge X_2 = (X_1 \cap X_2)$$
$$Join: X_1 \vee X_2 = (X_1 \cup X_2)$$

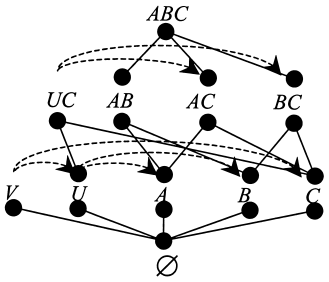


Fig. 2 Relationships on generalized itemsets (a part).

3.2 Ancestor-Descendant Relationship: *k*-Generalized Itemset Taxonomies

Definiton 3 (Ancestor-Descendant relationship): Let a binary relation $\delta_{\mathcal{A}} \subseteq \mathcal{P}(I) \times \mathcal{P}(I)$ be the *ancestor-descendant relationship*. For any $X_1, X_2 \in I_G$, $X_1 \delta_{\mathcal{A}} X_2$ can be denoted when X_2 is obtained by replacing one or more items in X_1 with one of their descendants, X_1 is called an *ancestor itemset* of X_2 (and X_2 is called a *descendant itemset* of X_1).

By using ancestor-descendant relationship, we can extend the original taxonomy (*1*-generalized itemset taxonomy) to express the ancestor-descendant relationships among *k*-length generalized itemsets.

Definiton 4 (*k*-generalized itemset taxonomy): The *k*-generalized itemset taxonomy is the partial order specified by an ancestor-descendant relationship $\delta_{\mathcal{A}}$ among generalized itemsets with the same *k*-length .

3.3 Combining Two Relationships

The generalized itemsets can be shown in a complex view that combines both subset-superset and ancestor-descendant relationships. For example, assume the taxonomy as in Fig. 1 (C) and a set of items $\{A, B, C, U, V\}$, the relationships among generalized itemsets are shown in Fig. 2. The solid lines express the subset-superset relationship where the lower itemset is a subset of the upper itemset, and the dashed lines express the ancestor-descendant relationship where the itemset at the beginning of an arrow is an ancestor itemset of the itemset at the end of the arrow.

4. Constraints on Generalized Itemsets

We can exploit these two relationships as constraints for efficient finding GFIs. Two lemmas are presented to justify the optimization as follows.

Lemma 1 (Subset-Superset Constraint): For any $X \in I_G$, if a generalized itemset X is frequent, all subsets of X are frequent. In conversely, if a generalized itemset X is infrequent, all supersets of X are infrequent.

Proof: Let $X, Y, Z \in I_G$ and $Z = X \cup Y$. 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. $\sigma(X)$ and $\sigma(Y)$. Thus, if Z satisfies minsup (frequent), both X and Y do too. If both or either of X and Y does not satisfy minsup (infrequent), then neither does Z. □

For example, given minsup = 67%, a generalized itemset ACD ($\sigma(ACD) = 33\%$) is infrequent. The superset of ACD, such as $ACDE$ ($\sigma(ACDE) = 33\%$) or $ABCDE$ ($\sigma(ABCDE) = 17\%$), are also infrequent. This constraint shows that we need not to consider the supersets of infrequent itemsets.

Lemma 2 (Ancestor-Descendant Constraints): For any $X \in I_G$ where $\hat{X}$ is an ancestor itemset of X, if X is frequent, then $\hat{X}$ is also frequent. In conversely, if $\hat{X}$ is infrequent, X is also infrequent.

Proof: Let $x, \hat{x} \in I$ and $Y, Z, \hat{Z} \subseteq I$. Assume that $Z = x \cup Y$, $\hat{Z} = \hat{x} \cup Y$. $\hat{x}$ is an ancestor 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 $t(x) \subseteq t(\hat{x})$. Thus, If Z satisfies minsup (frequent), so does $\hat{Z}$. If $\hat{Z}$ does not satisfy minsup (infrequent), neither does Z. □

For example, given minsup = 83%. A generalized itemset UE ($\sigma(UE) = 67\%$) is infrequent. The descendant itemsets of UE, such as AE ($\sigma(AE) = 33\%$) and BE ($\sigma(BE) = 33\%$), are also infrequent. This constraint shows that we need not to consider the descendant itemsets of infrequent itemsets.

5. Generalized Closed Itemsets

In this section, the concept of generalized closed itemsets is defined by extending the traditional concept of closed itemsets in ARM ([12], [18]) to cope with the generalized itemsets. We also show that a set of generalized closed frequent itemsets is sufficient to be the representative of a larger set of GFIs. In order to understand the generalized closed frequent itemset, we introduce a maximal generalized itemset which is another representation of a generalized itemset.

5.1 Maximal Generalized Itemsets

In general, a generalized itemset can be transformed into another representation that includes both original items and all of their ancestors. This representation, we call a maximal generalized itemset of a generalized itemset. The formal definition is stated as follows.

Definiton 5 (Maximal Generalized Itemset): Let $X \subseteq I$, X is called a *maximal generalized itemset* iff the following condition is satisfied $\forall i \, (i \in X \rightarrow \mathcal{ANC}(i) \subset X)$.

In every situation, each generalized itemset can always be transformed to each maximal generalized itemset and vice versa. Using the extended database, a generalized itemset can easily be transformed into the form of a maximal generalized itemset. This form is useful for finding generalized closed itemsets, since the concept of a closure finds a

maximal superset of an itemset that supports the same tidset as a generalized itemset (described below). Thus, maximal generalized itemsets are generated instead of generalized itemsets when we find generalized closed itemsets. However each element in the set can be mapped to its corresponding generalized itemset.

5.2 Generalized Closed Itemset Concept

Definiton 6 (Galois Connection): Let $X \subseteq I$, and $Y \subseteq T$. Then the mapping functions,

$$t : \mathcal{P}(I) \mapsto \mathcal{P}(T),\ t(X) = \{y \in T | \forall x \in X, x\delta y\}$$
$$i : \mathcal{P}(T) \mapsto \mathcal{P}(I),\ i(Y) = \{x \in I | \forall y \in Y, x\delta y\}$$

define a Galois connection between the power set of I and the power set of T.

The following properties hold for all $X, X_1, X_2 \subseteq I$ and $Y, Y_1, Y_2 \subseteq T$:

1. $X_1 \subseteq X_2 \rightarrow t(X_1) \supseteq t(X_2)$.
2. $Y_1 \subseteq Y_2 \rightarrow i(Y_1) \supseteq i(Y_2)$.
3. $X \subseteq i(t(X))$ and $Y \supseteq t(i(Y))$.

The mapping $t(X)$ is the maximal tidset which contains the generalized itemset X , given by $t(X) = \bigcap_{x \in X} t(x)$. The mapping $i(Y)$ is the maximal generalized itemset which is contained in the tidset Y, given by $i(Y) = \bigcap_{y \in Y} i(y)$. For example, $t(UDE) = t(U) \cap t(D) \cap t(E) = 123456 \cap 13456 \cap 1356 = 1356$, and $i(356) = i(3) \cap i(5) \cap i(6) = VUBCWDE \cap VUABCWDE \cap VUBCWDE = VUBCWDE$.

Apart from the closure operator, the generalized closure operator is defined as follows:

Definiton 7 (Generalized Closure): Let $X \subseteq I$, and $Y \subseteq T$. The two mappings

$$gc_{it} : \mathcal{P}(I) \mapsto \mathcal{P}(I),\ gc_{it}(X) = i \circ t(X) = i(t(X))$$
$$gc_{ti} : \mathcal{P}(T) \mapsto \mathcal{P}(T),\ gc_{ti}(Y) = t \circ i(Y) = t(i(Y))$$

are generalized closure operators on generalized itemset and tidset respectively. X is called a *generalized closed itemset* (GCI) when $X = gc_{it}(X)$, and Y is called a *generalized closed tidset* (GCT) when $Y = gc_{ti}(Y)$.

For $X \subseteq I$ and $Y \subseteq T$, the generalized closure operators gc_{it} and gc_{ti} satisfy the following properties:

1. $Y \subseteq gc_{ti}(Y)$.
2. $X \subseteq gc_{it}(X)$.
3. $gc_{it}(gc_{it}(X)) = gc_{it}(X)$, and $gc_{ti}(gc_{ti}(Y)) = gc_{ti}(Y)$.

The first property tells that Y is a subset of its generalized closure. For example, let $Y = 135$, $gc_{ti}(135) = t(i(135)) = t(UCDE) = 1356$. Since $135 \neq gc_{ti}(135) = 1356$, such that 1356 is a generalized closed tidset while 135 is not. The second property says that X is a subset of its generalized closure. For example, $gc_{it}(VWDE) = i(t(VWDE) = i(1356) = VUCWDE$. Since $VWDE \neq gc_{it}(VWDE) = VUCWDE$, such that $VUCWDE$ is a GCI while $VWDE$ is not. Note that each GCI is a maximal generalized itemset, but it can be mapped to a generalized itemset. From the previous example, $VWDE$ and $VUCWDE$ can be transformed to the generalized itemsets VDE and $UCDE$, respectively. In generalized itemset form, this means that the GCI of VDE is $UCDE$. The last property says that the round-trip of mapping will obtain the same closure.

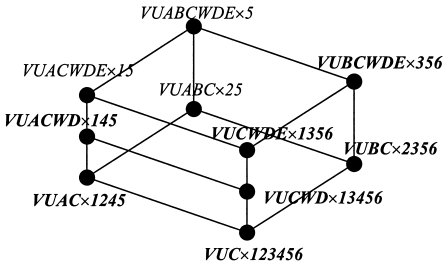


Fig. 3 Galois lattice of concepts and frequent concepts.

For any GCI X, there exists a companion GCT Y, with the property of $Y = t(X)$ and $X = i(Y)$. Such a GCI and GCT pair $X \times Y$ is called a *concept*. All possible concepts can form a Galois lattice of concepts as shown in Fig. 3.

5.3 Generalized Closed Frequent Itemsets

The support of a concept $X \times Y$ is a percentage of the size of closed tidset Y to the total number of transactions ($|Y|/|T|$). A GCI is called a *generalized closed frequent itemset* (GCFI) when its support is at least minsup.

Lemma 3 (Equivalence of Support): For any generalized itemset X, its support is equal to the support of its generalized closure ($\sigma(X) = \sigma(gc_{it}(X))$).

Proof: The support of X, $\sigma(X)$ is $|t(X)|/|T|$, and the support of $gc_{it}(X)$, $\sigma(gc_{it}(X))$ is $|t(gc_{it}(X))|/|T|$. To prove the lemma, we have to show that $t(X) = t(gc_{it}(X))$.

Since gc_{ti} is a generalized closure operator, it satisfies the first property that $t(X) \subseteq gc_{ti}(t(X)) = t(i(t(X))) = t(gc_{it}(X))$. Thus $t(X) \subseteq t(gc_{it}(X))$. In the other case, the $gc_{it}(X)$ provides the maximal itemset, i.e. $X \subseteq gc_{it}(X)$, which implies that $t(X) \supseteq t(gc_{it}(X))$ due to the first property of Galois connection. Thus we conclude that $t(X) = t(gc_{it}(X))$. □

Implicitly, the lemma states that all GFIs can be uniquely determined by the GCFIs since the support of any generalized itemsets will be equal to its generalized closure. Given a set of GCFIs, a Hasse diagram representing the subset-superset relationship among concepts in the Galois lattice, can be constructed using the method in [16] with $O(l.m.w(\mathcal{L}).d(\mathcal{L}))$ in time, where l is the average size of generalized itemsets, m is the number of items, $w(\mathcal{L})$ is the width of the lattice and $d(\mathcal{L})$ is the maximal degree of a lattice node. Consequently, all GFIs and their supports can be efficiently determined from the GCFIs and their Hasse diagram (Galois lattice). However, all GFIs need not to be discovered, since a set of GCFIs is typically used to construct a minimal set of non-redundant rules as shown in [3]

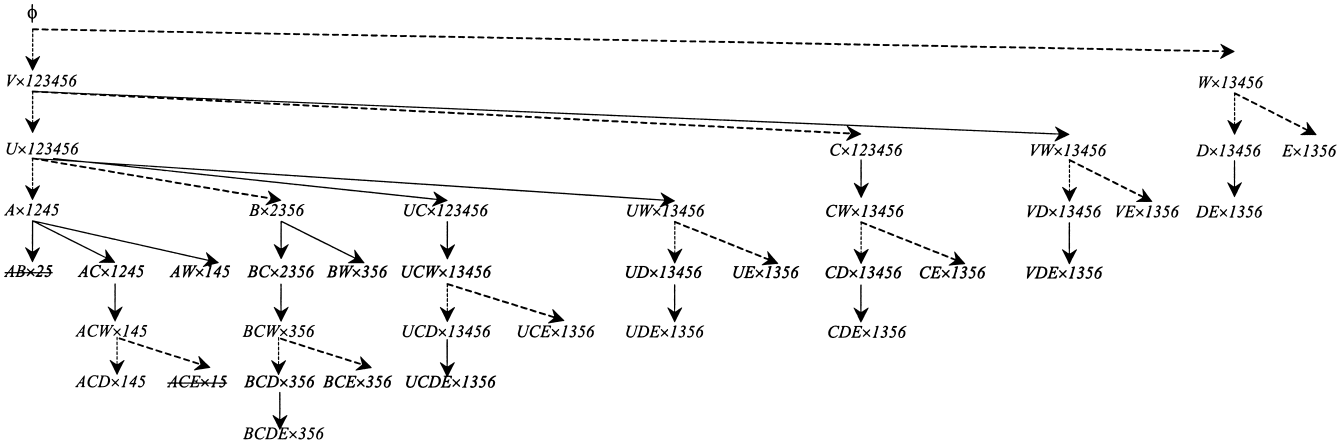


Fig. 4 Set enumeration using *SET* algorithm (minsup = 50%).

and [17]. In the worst case, the number of GCFIs is equal to the number of GFIs, but typically it is much smaller. From our example, there are 10 GCIs which are the representatives of a large amount of all generalized itemsets as shown in Fig. 3. With minsup = 50%, only 7 concepts (in bold font) are GCFIs.

6. Algorithms: *SET* and *cSET*

This section describes two algorithms, *SET* and *cSET*, that utilize two constraints for efficiently mining GFIs and GCFIs, respectively. For fast finding all GFIs, each of the lemma in Sect. 4 can be applied to each relationship of generalized itemsets. Lemma 1 can be applied to the lattice of generalized itemsets while Lemma 2 can be applied to the taxonomies of k-generalized itemsets. Lemma 1 concerns with the subset-superset relationship which exists in the generalized itemset lattice, while Lemma 2 concerns with the ancestor-descendant relationship which exists in the taxonomies of k-generalized itemsets. These lemmas enable us to avoid generating itemsets that are dominantly infrequent. To enumerate all GFIs, we can traverse each relationship of generalized itemsets. For bi-directional traversal, the lattice of generalized itemsets should be traversed from subsets to their supersets and from ancestor itemsets to their descendant itemsets. Before generating any generalized itemsets, all of their subsets must be frequent. Similarly, an ancestor itemset must be frequent before generating its descendant itemsets. Following these approaches, only supersets and descendant itemsets of GFIs are generated.

6.1 *SET* Algorithm

Most of computational cost on generating all GFIs is to count supports of the generalized itemsets for checking whether they are frequent or not, and checking to eliminate meaningless itemsets. The *SET* algorithm applies two techniques for enumerating GFIs using an extended vertical database format. The first one is to apply our novel set enumeration to generate only generalized itemsets without intensive checking on meaningless itemsets. This set enumeration was proposed in our previous work ([15]). The second technique is to apply a bi-directional traversal during set enumeration in order to avoid generating obvious infrequent itemset.

As stated in Sect. 5.1, a generalized itemset is transformed from a maximal generalized itemset by omitting the ancestors of items in the maximal set. However, the representation of a maximal generalized itemset is useful for describing the process of set enumeration as follows. Normally, two itemsets can be joined together when they have the same size k and contain the preceding $k-1$ itemset for avoiding redundant enumeration. Among maximal generalized itemsets, the join can be produced by a set union. For example, joining VUA with $VUC = VUAC$. However, when reducing to the generalized itemset, the join can be produced by a set union of the first itemset with the last item of the second itemset. For example, joining A with $UC = A \cup C = AC$ where its tidset is given by a set intersection. This join operation on generalized itemsets is used in the *SET* algorithm.

For clarity, we explain the *SET* algorithm by the example illustrated in Fig. 4. With minsup = 50%, the proposed set enumeration starts with an empty set. Then, all second-leveled items of the taxonomy which are frequents, i.e. V and W, are added to the second level of the tree. The children under each generalized itemset are generated in two manners. First, we generate *taxonomy-based child itemsets* (based on ancestor-descendant relationship) by replacing the right most item of that generalized itemset with one of their children (if any). Secondly, we generate all *join-based child itemsets* (based on subset-superset relationship) by union that generalized itemsets with the last item of their siblings that have higher orders. For example, generating the children of itemset V, we first generate taxonomy-based child itemsets, i.e. U and C, and then generate join-based child itemsets, i.e. VW. Each generalized itemset which is generated must be frequent, otherwise it will be pruned. With

the same approach, the process recursively occurs until no new GFI is generated. Finally, a complete GFI tree is constructed without excessive checking cost. All generalized itemsets in the Fig. 4, except ones with a cross, are GFIs. The pseudo-codes of *SET* will be shown in Sect. 6.3.

6.2 *cSET* Algorithm

This section presents an extension of *SET* algorithm, called *cSET* algorithm, for mining GCFIs. Since the GCFI is in the form of the maximal generalized itemset, we thus intend to enumerate the generalized itemsets in the form of maximal generalized itemsets. The same process of set enumeration in *SET* for bi-directional traversal is used, but the join operation is given by a set union on maximal generalized itemsets with some conditional properties to discard non-GCFIs.

In the process of set enumeration, the following conditional properties are used to reduce the number of GCFIs needed to be generated. Assume that $X_1 \times t(X_1)$ is joined with $X_2 \times t(X_2)$:

1. If $t(X_1) = t(X_2)$, then (1) replace every occurrence of X_1 with $X_1 \cup X_2$, (2) remove X_2 if X_2 is a sibling of X_1, and (3) generate taxonomy-based child itemsets of the current new X_1 (since the former X_1 is replace by $X_1 \cup X_2$).
2. If $t(X_1) \subset t(X_2)$, then (1) replace every occurrence of X_1 with $X_1 \cup X_2$, and (2) generate taxonomy-based child itemsets of the current new X_1.
3. If $t(X_1) \supset t(X_2)$ and $t(X_1) \cap t(X_2)$ is not contain in hash table, then (1) store $t(X_1) \cap t(X_2)$ in the hash table, (2) remove X_2 if X_2 is a sibling of X_1, and (3) generate $X_1 \cup X_2$ under X_1 in tree.
4. If $t(X_1) \neq t(X_2)$ and $t(X_1) \cap t(X_2)$ is not contain in hash table, (1) store $t(X_1) \cap t(X_2)$ in the hash table, and (2) generate $X_1 \cup X_2$ under X_1 in tree.

For clarity, we explain the *cSET* algorithm using the example in Fig. 5. With minsup = 50%, the *cSET* algorithm starts with an empty set. Then, all second-leveled items of the taxonomy which are frequent, i.e. *V* and *W*, are added to the second level of tree. Similar to *SET*, children are generated based on two manners but the form of an itemset has changed to be the maximal generalized itemset. The taxonomy-based child itemset is generated by a set union between the current itemset and one of the children of the rightmost item in that set according to taxonomy (if any). The join-based child itemset is normally generated by a set union on maximal generalized itemsets as previously described in Sect. 6.1. The first child of *V* from taxonomy (i.e. *U*) produces taxonomy-based itemset VU. The first property holds for VU, which results in replacing *V* with VU and then generating its taxonomy-based itemsets, i.e VUA and VUB with the fourth property. The second child of *V* from taxonomy (i.e. *C*) is still joined with the current itemset (VU), which produces VUC. Again, the first property holds for VUC, which results in replacing VU and the children in the tree under VU with VUC. Because of this, VUA and VUB are replaced by $VUCA$ and $VUCB$, respectively. Next, the current itemset (VUC) joins with its sibling (*W*), i.e. $VUCW$. The third property holds for $VUCW$, which results in removing *W* and then generating $VUCW$ under the current VUC. This process shows that the first and third properties can help us to discard some itemsets and the subtrees under those itemsets which are clearly not to be GCFIs. That is *W* and the subtree under W are pruned. With the same approach, the process recursively occurs until no new GCFI is generated. The hash tree is used for checking whether the current tidset occurs in the previous enumeration or not. Instead of 36 GFIs in Fig. 4, we enumerate only 7 GCFIs as shown in Fig. 5. This action results in reducing computational time. All remaining maximal generalized itemsets in Fig. 5, except ones with a cross, are GCFIs.

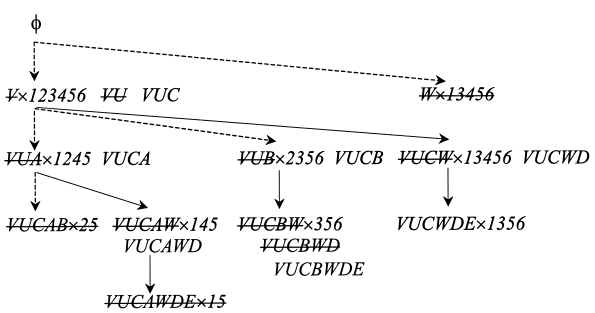


Fig. 5 Set enumeration using *cSET* algorithm (minsup = 50%).

6.3 Pseudo-Codes Description

The pseudo-codes of *SET* and its extension *cSET* are shown in Fig. 6. For the *SET* algorithm, line 11.c, 13.c, 17.c and 18.c–30.c are ignored. In the main procedure *SET-Main*, the subordinate function called *SET-Extend*, recursively creates a subtree using the proposed method. The *GenTaxChild* function produces a taxonomy-based child itemset while the *GenJoinChild* function produces a join-based child itemset. In line 12 and 16, the generated itemsets must be checked to ensure whether they are frequent or not. The *NewNode* function generates a new itemset under the current itemset. For the *cSET* algorithm, line 11.s, 13.s and 17.s are ignored. Similar to *SET*, *cSET* uses *SET-Main*, *SET-Extend*, *GenTaxChild* and *GenJoinChild* as well as *cSET-Property* and *Hash* functions. Since the form of a generalized itemset in *SET* is generalized itemset but *cSET* is maximal generalized itemset, line 13 of *SET* and *cSET* are different. Instead of *NewNode* function in line 13.s and 17.s, we use the *cSET-Property* function in line 13.c and 17.c to check the conditional properties as previously described in Sect. 6.2 for generating only GCFIs. The *Hash* function is used for checking whether the current tidset occurs in the previous enumeration or not by returning 1 when exists, or storing that tidset in the hash table and return 0 when not exists. Following the *SET* algorithm, we will get the tree of all GFIs while the *cSET* algorithm will get the tree of all GCFIs.

```
SET-Main(Database,Taxonomy,minsup):
1.  Root = Null Tree
2.  For each x ∈ Second-level Items of Taxonomy
3.      If |t(x)| ≥ minsup then NewNode(Root, x)
4.  SET-Extend(Root)

SET-Extend(Node)    //Recursively generate tree in depth-first
5.  For each Fi ∈ Node.Child
6.      Fi.Child = NULL
7.      GenTaxChild(Fi)
8.      GenJoinChild(Fi)
9.      If Fi.Child != NULL then SET-Extend(Fi)

GenTaxChild(Fi)     //Generate Taxonomy-based Child Itemsets
10. For each x ∈ Child of LastItem(Fi) in Taxonomy
11.s    C = Fi after Replace LastItem(Fi) with x    //For SET
11.c    C = Fi ∪ x                                  //For cSET
12.     If |t(C)| ≥ minsup then
13.s        NewNode(Fi, C)                          //For SET
13.c        cSET-Property(Fi, x, C)                 //For cSET

GenJoinChild(Fi)    //Generate Join-based Child Itemsets
14. For each Fj ∈ Sibling of (Fi)                   // j > i
15.     C = Fi ∪ LastItem(Fj)
16.     If |t(C)| ≥ minsup then
17.s        NewNode(Fi, C)                          // For SET
17.c        cSET-Property(Fi, Fj, C)                //For cSET

cSET-Property(Fi, Fj, C)        //4 Properties for cSET
18.c if t(Fi) = t(Fj) then                          //Prop.1
19.c     Replace all Fi with C;
20.c     if Fj ∈ Sibling of Fi then Remove Fj;
21.c     GenTaxChild(Fi);
22.c else if t(Fi) ⊂ t(Fj) then                     //Prop.2
23.c     Replace all Fi with C;
24.c     GenTaxChild(Fi);
25.c else if t(Fi) ⊃ t(Fj) then                     //Prop.3
26.c     if Fj ∈ Sibling of Fi then Remove Fj;
27.c     if !Hash(t(C)) then NewNode(Fi, C);
28.c else if !Hash(t(C)) then NewNode(Fi, C)        //Prop.4

Hash(tidset)                    //Find tidset in Hash Table
29.c if Found tidset in Hash Table then return 1
30.c else Add tidset in Hash Table; return 0.
```

Fig. 6 The pseudo-codes of *SET* and *cSET* algorithm.

7. Experimental Results

For testing the performance of our approaches, we compare *SET* and *cSET* algorithms with the two popular algorithms, i.e. Cumulate ([14]) and Prutax ([8]). All algorithms were coded in C. Experiments were done on a 1.7 GHz Pentium IV PC with 640 MB of main memory, running Windows 2000.

To measure the exact execution time of algorithms (excluding intensive I/O cost), we make the dataset and its taxonomy reside on the memory. Therefore, the memory size should be large enough to store the data in order to avoid page swapping time. Here, we can illustrate the calculation of memory needed for the largest dataset as follows. The largest dataset in the experiments is the synthetic dataset with 1 million (10^6) transactions which contains 10 items per transaction and 5 fanouts per item. Converting the dataset to the vertical format, where an item is encoded to an integer, the required memory for this dataset is $10^6 \times 10 \times 5 \times 4$ bytes $\approx$ 200 MB. From our investigation, the exact memory usage for this dataset including other variables of the program and required memory for operating system is at least 32 MB. Approximately, 300 MB are required where the remaining memory space can be used for computation process. To process more transactions, we need more memory. Anyway, 640 MB is large enough for the current experiments.

Table 1 The default value of parameters in synthetic datasets.

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

$$\text{Depth-ratio} = \frac{\text{probability that item in a rule comes from level i}}{\text{probability that item comes from level i + 1}}$$

Table 2 The real datasets and their parameters.

Dataset	Parameters			
	#Trans	#Items*	#Roots	Fanout
MushroomR40F3	8124	159	40	3
MushroomR24F5	8124	143	24	5
ChessR15F5	3196	90	15	5

*#Items include both leaf (original) items and non-leaf items.

7.1 Datasets

The synthetic and real datasets are used as benchmarks for evaluating the performance. The synthetic datasets were automatically generated by a generator tool[†]. They mimic the transactions in a retailing environment. The important default values of parameters in synthetic datasets are shown in Table 1. Two standard real datasets, i.e. mushroom and chess[††], are also used for investigating our methods in actual environment. These real datasets are often used for testing the performance of data mining algorithms. There is no taxonomy specified in the original real datasets. Therefore, we construct an additional taxonomy for each dataset by defining a number of roots and fanout of the taxonomy in order to make all original items appear in the leaf level of the taxonomy. All 119 original items of mushroom can be covered in the second depth of a taxonomy with the number of roots and fanout respectively be 40 and 3 (or 24 and 5), and 75 original items of chess can be covered in the second depth of taxonomy with the number of roots and fanout respectively be 15 and 5. These taxonomies are suitable since the original real datasets we used are dense and many items usually appear in the most portions of transactions. Therefore with higher fanout, the excessive amount of dense patterns may occur and then the algorithms suffer with the memory limitation problem. Thus, we get three different datasets as shown in Table 2. These three real datasets are fixed

[†]The generator tool are provided by IBM Almaden Site.

[††]Original mushroom and chess are provided by UCI Machine Learning Database Repository.

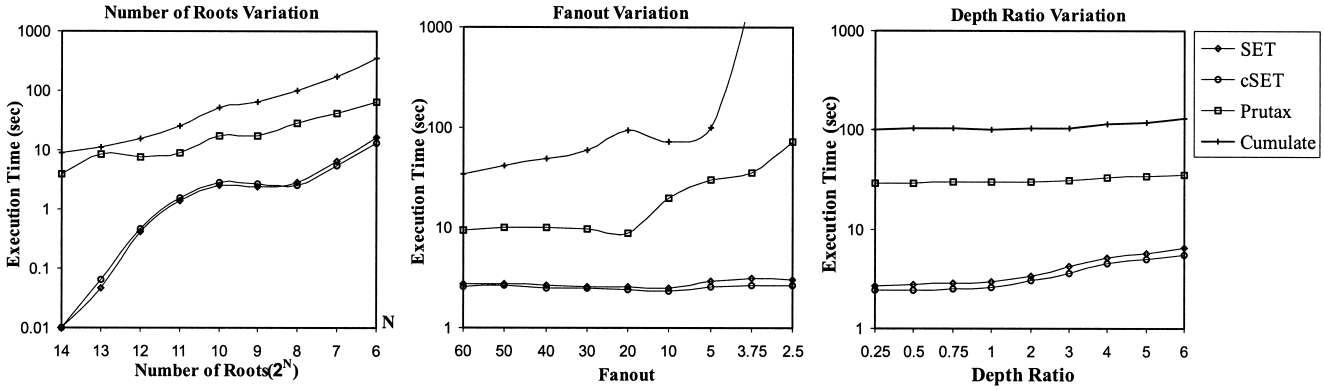


Fig. 7 Experimental results: taxonomy characteristics.

throughout the experiments.

7.2 Performance Testing

Four experiments are performed to investigate the performance of the algorithms in different situations. At first, we study on how the algorithms perform on different characteristics of taxonomy. Secondly, we investigate the performances of the algorithms on different scaling of database. Thirdly, the performance of algorithms with various minsups is evaluated, and the numbers of frequent patterns (i.e. GFIs and GCFIs) are compared. Finally, the memory usage of each algorithm is checked using both synthetic and real datasets. We only use real datasets in the third and fourth experiments since it is not possible to vary the characteristics of their taxonomy.

Taxonomy Characteristics: Figure 7 shows the execution time of algorithms when the characteristics of taxonomy are changed. The performances of *SET* and *cSET* are so close since the numbers of GFIs and GFCIs almost equal (shown in the latter experiment). Both algorithms are approximately 4 to 180 times faster than Prutax and 22 to 230 times faster than Cumulate. In case of the smaller number of roots, taxonomy levels become deeper and then the number of ancestor itemsets turns to be larger. *SET* and *cSET* are not sensitive to this situation while Prutax requires more time for checking and Cumulate needs more time to modify the transactions. With different fanouts, the number of children of each non-leaf item in taxomomy is varied. The number of ancestor itemsets in lower fanout is larger than higher fanout. As shown in the figure, decreasing the fanout has an effect similar to decreasing the number of roots. For a lower depth ratio, we gain more frequent patterns that contain items coming from the lower parts rather than the upper parts of taxonomy. The number of ancestor itemsets increases and this phenomenon results in more time consuming in Prutax and Cumulate. *SET* and *cSET* achieve approximately 6–10 times faster than Prutax and 20–38 times faster than Cumulate with depth-ratio variation.

Scaling Database: Figure 8 shows the execution time of each algorithm when the database is scaled up and down. In this experiment, all taxonomy parameters are fixed to their default values, but only the number of transactions and the number of items are scaled. We observe an exponential increment in the running time with the increasing number of transactions. However, *SET* and *cSET* still perform well with the large number of transactions. With the scaling number of items, *SET* and *cSET* are not affected by this variation since an item occurs sparsely in the transactions and then the number of GFIs to be counted is reduced, but it results in more time consuming in Prutax and Cumulate.

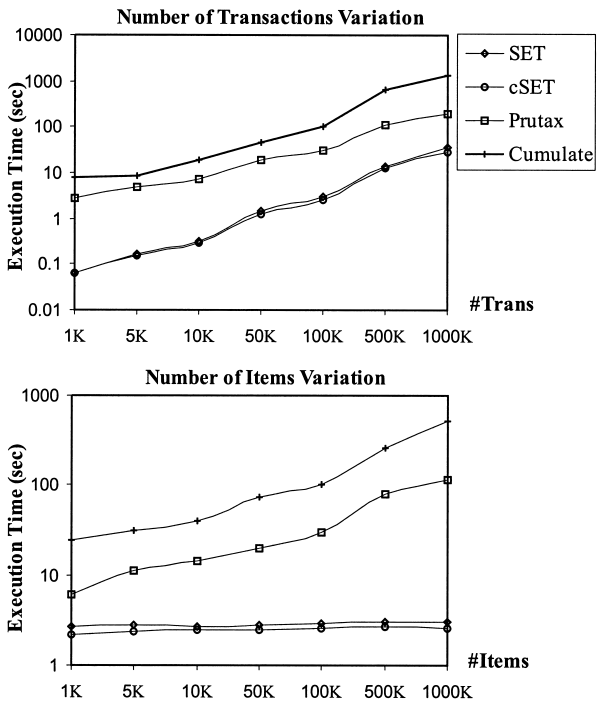


Fig. 8 Experimental results: scaling database.

Minsup Variation and #Frequent Patterns: Typically, the real datasets are very dense, i.e. frequent patterns are mostly long even high value of minsup while the synthetic datasets are sparse. Table 3 shows the execution time of each algorithm and the number of frequent patterns, when minsup is varied. The execution time is exponential growth

Table 3 Experimental results: Minimum support variation and number of frequent patterns.

Minsup	Execution Time (sec)				#Frequent Patterns	
	SET	*cSET*	Prutax	Cumulate	#GFIs	#GFCIs
Dataset: SynR250F5D1						
4	0.7	0.7	9.5	30.6	228	226
3	1.0	1.0	11.3	37.7	404	401
2	1.5	1.4	14.9	58.5	848	843
1.5	2.0	1.8	19.3	73.2	1,484	1,475
1	2.9	2.6	29.9	101.1	3,235	3,211
0.75	4.0	3.3	40.8	71563.6	5,684	5,633
Dataset: MushroomR40F3						
100	0.03	0.01	0.03	0.39	95	1
90	0.06	0.02	0.06	0.69	431	5
80	0.31	0.02	0.38	1.58	1,839	18
70	1.09	0.05	1.16	3.19	7,983	42
60	2.92	0.05	2.98	8.59	19,543	102
50	9.48	0.13	9.69	57.09	68,095	297
Dataset: MushroomR24F5						
100	0.14	0.01	0.05	0.52	191	1
90	1.41	0.03	1.23	3.60	9,023	34
80	4.59	0.06	4.00	14.19	28,127	84
70	12.88	0.13	12.27	79.77	85,343	216
60	29.94	0.28	29.14	360.77	204,143	485
50	76.67	0.59	78.36	2141.99	524,231	1,102
Dataset: ChessR15F5						
100	2.36	0.01	2.64	11.08	32,767	1
98	15.41	0.01	20.03	401	231,423	22
96	25.45	0.02	158.14	1127.92	389,119	45
94	65.69	0.02	926.25	7081.02	935,935	125
92	110.75	0.05	2895.33	19334.05	1,602,559	270
90	179.53	0.06	7120.33	48890.14	2,565,631	499

with decreasing minsup. *SET* and *cSET* provide similar performance in the synthetic dataset (SYNR250F5D1), but they perform differently on the real datasets (i.e. MushroomR40F3, MushroomR24F5 and ChessR15F5). Cumulate cannot be executed with a lower minsup in a synthetic dataset since it generates a lot of candidates which are at last infrequent. In the real datasets, the performances of *SET* and Prutax are quite close since the sizes of real datasets are small, resulting in a trivial hashing time for Prutax. However, *cSET* still performs better than other algorithms, since the number of GCFIs is excessively smaller than the number of GFIs as shown in Table 3. We observe that the difference between the number of GFIs and GCFIs is much smaller in the synthetic datasets but dominantly larger in the real datasets.

Memory Usage: Table 4 shows the maximum memory usage of each algorithm with different minsups in the synthetic and real datasets. For the synthetic dataset, the memory usage of *cSET* is trivially greater than *SET* since the number of their frequent patterns are almost equal and *cSET* has to hold some GFIs in memory for checking. However, their memory usage are rather smaller than Prutax. For the real datasets, the memory usage of *SET* and *cSET* are smaller than the other two algorithms, but the memory usage of Cumulate grows excessively since a lot of candidates are generated and held in memory. These results confirm that *SET* and *cSET* are superior to the other algorithms in memory utilization.

Table 4 Maximum memory usage of each algorithms.

Dataset (%Minsup)	Maximum Memory Usage (MB)			
	SET	*cSET*	Prutax	Cumulate
SynR250F5D1 (2%)	35.9	44.6	52.8	28.2
SynR250F5D1 (1%)	46.2	48.9	55.0	136.3
MushroomR40F3 (80%)	12.9	11.6	16.3	23.4
MushroomR40F3 (60%)	13.2	13.7	17.4	29.8
MushroomR24F5 (80%)	13.0	13.4	16.4	29.4
MushroomR24F5 (60%)	13.8	22.3	17.8	79.9
ChessR15F5 (100%)	10.5	9.4	13.9	35.9
ChessR15F5 (98%)	10.8	9.8	14.9	72.4
ChessR15F5 (96%)	10.9	10.4	15.2	138.4

8. Conclusions

In this work, we presented a theoretical framework of generalized itemsets based on subset-superset relationship (represented by lattice of generalized itemsets), and ancestor-descendant relationship (represented by taxonomy of k-generalized itemsets). To efficiently discover all generalized frequent itemsets, we introduced two constraints corresponding to these two relationships. We proposed *SET* and *cSET* algorithms to enumerate generalized frequent itemsets and generalized closed frequent itemsets, respectively. *SET* and *cSET* use an efficient traversal on the combination of two relationships to avoid generating meaningless itemsets, and apply two constraints to prevent counting useless generalized itemsets that are obviously infrequent. This makes *SET* and *cSET* efficiently find all frequent patterns. A number of experiments showed that *SET* and *cSET* outperform the previous well-known algorithms in both computational time and memory utilization, especially for real situations. There are other problems related to ARM to be considered in GARM, including incremental data mining, constraint-based mining, interesting measures, negative rule mining, parallel mining, and so on. They are left as our further explorations.

Acknowledgement

This paper has been supported by Thailand Research Fund (TRF) and NECTEC under project number NT-B-06-4C-13-508.

References

[1] R. Agrawal, T. Imielinski, and A.N. Swami, "Mining association rules between sets of items in large databases," Proc. 1993 ACM SIGMOD International Conference on Management of Data, ed. P. Buneman and S. Jajodia, pp.207–216, Washington, D.C., May 1993.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," Proc. 20th International Conference on Very Large Data Bases, VLDB, ed. J.B. Bocca, M. Jarke, and C. Zaniolo, pp.487–499, Morgan Kaufmann, 1994.

[3] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal, "Mining minimal non-redundant association rules using frequent closed itemsets," Lecture Notes in Computer Science, vol.1861, pp.972–986, 2000.

[4] B.A. Davey and H.A.Priestley, Introduction to Lattices and Order, Second ed., Cambridge University Press, 2002.
[5] B. Ganter and R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer-Verlag, New York, 1997.
[6] J. Han and Y. Fu, "Mining multiple-level association rules in large databases," Knowledge and Data Engineering, vol.11, no.5, pp.798–804, 1999.
[7] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufman, 2001.
[8] J. Hipp, A. Myka, R. Wirth, and U. Güntzer, "A new algorithm for faster mining of generalized association rules," Proc. 2nd European Conference on Principles of Data Mining and Knowledge Discovery (PKDD '98), pp.74–82, Nantes, France, Sept. 1998.
[9] S.-Y. Hwang and E.-P. Lim, "A data mining approach to new library book recommendations," Lecture Notes in Computer Science ICADL 2002, pp.229–240, Singapore, Dec. 2002.
[10] C.L. Lui and F.L. Chung, "Discovery of generalized association rules with multiple minimum supports," Proc. 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000), pp.510–515, Lyon, France, Sept. 2000.
[11] A. Michail, "Data mining library reuse patterns using generalized association rules," International Conference on Software Engineering, pp.167–176, 2000.
[12] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," Lecture Notes in Computer Science, vol.1540, pp.398–416, 1999.
[13] T. Shintani and M. Kitsuregawa, "Parallel mining algorithms for generalized association rules with classification hierarchy," Proc. 1998 ACM SIGMOD International Conference on Management of Data, pp.25–36, 1998.
[14] R. Srikant and R. Agrawal, "Mining generalized association rules," Future Gener. Comput. Syst., vol.13, no.2/3, pp.161–180, 1997.
[15] K. Sriphaew and T. Theeramunkong, "A new method for fiding generalized frequent itemsets in generalized association rule mining," Proc. Seventh International Symposium on Computers and Communications, ed. A. Corradi and M. Daneshmand, pp.1040–1045, Taormina-Giardini Naxos, Italy, July 2002.
[16] P. Valtchev, R. Missaoui, and P. Lebrun, "A fast algorithm for building the Hasse diagram of a Galois lattice," Colloque LaCIM2000, Combinatoire, Informatique et Applications, pp.293–306, Sept. 2000.
[17] M.J. Zaki, "Generating non-redundant association rules," Proc. 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.34–43, Boston, MA, Aug. 2000.
[18] M.J. Zaki and C.-J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining," Proc. Second SIAM International Conference on Data Mining, ed. R. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, pp.459–473, Arlington, VA, April 2002.
[19] M.J. Zaki and M. Ogihara, "Theoretical foundations of association rules," Proc. 1998 ACM SIGMOD International Conference on Management of Data, pp.7:1–7:8, Seattle, WA, June 1998.
[20] C. Zhang and S. Zhang, "Association rule mining: Models and algorithms," Lecture Notes in Artificial Intelligence, Springer-Verlag, July 2002.

Kritsada Sriphaew received the B.E. in Computer Engineering from King Mongkut's Institute of Technology Ladkrabang, Thailand in 2000. He is now a doctoral candidate in Information Technology Program, Sirindhorn International Institute of Technology, Thailand.

Thanaruk Theeramunkong recieved a bachelor degree in Electric and Electronics, and master and doctoral degrees in Computer Science from Tokyo Institute of Technology in 1990, 1992 and 1995, respectively. He manages a text data mining project funded by NECTEC, Thailand. His current research interests include data mining, machine learning, natural language processing, and information retrieval.