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 ([11], [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 bread, 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 ([11], [2]). For clarity, all explanations in the section are illustrated using an
example shown in Fig. 1.

Let $T$ be a taxonomy, a directed acyclic graph on items, which represents is-a relationship by edges (e.g. Fig. 1 (C)). The items in $T$ 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 $\hat{x} \in I$, $\hat{x}$ is an *ancestor of* $x$ (conversely $x$ is a *descendant of* $\hat{x}$) when there is a path from $\hat{x}$ to $x$ in $T$. For any $x \in I$, a set of all its ancestors (descendants) is denoted by $\text{ANC}(x)$ ($\text{DES}(x)$). For example, $\text{ANC}(B) = \{U, V\}$ and $\text{DES}(W) = \{D, E\}$.

A generalized itemset $G$ is an itemset each element of which is not an ancestor of the others, $G = \{i \in I | j \in G, i \notin \text{ANC}(j)\}$. For example, $\{A, B\}$ ($AB$ for short), $\{A, W\}$ ($AW$ for short) are generalized itemsets. Let $IG = \{G_1, G_2, \ldots, G_l\}$ be a finite set of all generalized itemsets. Note that, for $1 \leq i \leq l$, $G_i \subseteq I$ and $IG \subseteq P(I)$. 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 $G_1, G_2 \in IG$, $G_1 \cap G_2 = \emptyset$, 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 $\sigma(A \cup C)/\sigma(A) |T| = |\{1245\}/|6/6 = 4/6 or 67\%$ 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

**Definition 1** (Subset-superset relationship): Let a binary relation $\delta_S \subseteq P(I) \times P(I)$ be the *subset-superset relationship*. For any $X_1, X_2 \in IG$, $X_1\delta_S X_2$ is denoted when $X_1$ is a subset of $X_2$ ($X_2$ is a superset of $X_1$).

**Definition 2** (Lattice of generalized itemsets): The lattice of generalized itemsets is the partial ordered specified by a subset-superset relationship $\delta_S$, 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 $X_1, X_2 \in IG$,

- **Meet** $: X_1 \cap X_2 = (X_1 \cap X_2)$
- **Join** $: X_1 \vee X_2 = (X_1 \cup X_2)$
3.2 Ancestor-Descendant Relationship: \( k \)-Generalized Itemset Taxonomies

**Definition 3** (Ancestor-Descendant relationship): Let a binary relation \( \delta_{\mathcal{R}} \subseteq \mathcal{P}(I) \times \mathcal{P}(I) \) be the ancestor-descendant relationship. For any \( X_1, X_2 \in I_G \), \( X_1 \delta_{\mathcal{R}} X_2 \) 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.

**Definition 4** (\( k \)-generalized itemset taxonomy): The \( k \)-generalized itemset taxonomy is the partial order specified by an ancestor-descendant relationship \( \delta_{\mathcal{R}} \) 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 GFI. 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 \( Z \). \( \square \)

For example, given \( \text{minsup} = 67\% \), 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 \( X \) 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 \( Z \), \( \sigma(Z) = |t(Z)| = |t(x) \cap t(Y)| \) must be greater than or equal to the support of \( Z \), \( \sigma(Z) = |t(\hat{Z})| = |t(x) \cap t(Y)| \) 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 \( Z \). \( \square \)

For example, given \( \text{minsup} = 83\% \). 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 GFI. 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.

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

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

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

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

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

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 \( i(356) = i(3) \cap i(5) \cap i(6) = VUABCWDE \cap VUACWDE = VUACWDE \).

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

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

\[
gc_{ci} : \mathcal{P}(I) \mapsto \mathcal{P}(I), \ gc_{ci}(X) = i \circ t(X) = i(t(X))
\]

\[
gc_{it} : \mathcal{P}(T) \mapsto \mathcal{P}(T), \ gc_{it}(Y) = t \circ i(Y) = t(i(Y))
\]

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

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

1. \( Y \subseteq gc_{ci}(Y) \).
2. \( X \subseteq gc_{it}(X) \).
3. \( gc_{ci}(gc_{it}(X)) = gc_{ct}(X) \) and \( gc_{it}(gc_{ci}(Y)) = gc_{ct}(Y) \).

The first property tells that \( Y \) is a subset of its generalized closure. For example, let \( Y = 135, gc_{it}(135) = t(i(135)) = t(UCDE) = 1356 \). Since \( 135 \neq gc_{ci}(135) = 1356 \), such that \( 135 \) is a generalized closed tidset while \( 135 \) is not. The second property says that \( X \) is a subset of its generalized closure. For example, \( gc_{ci}(VWDE) = i(t(VWDE)) = i(1356) = VUCWDE \). Since \( VWDE \neq gc_{ci}(VWDE) = VUCWDE \), 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 \( UCD \), respectively. In general, itemset form, this means that the GCI of \( VDE \) is \( UCD \). The last property says that the round-trip of mapping will obtain the same closure.

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 \( \sigma(X) = \sigma(gc_{ci}(X)) \).

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

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

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 GFIs, a Hasse diagram representing the subset-superset relationship among concepts in the Galois lattice, can be constructed using the method in [16] with \( O(l.m.w(l).d(l)) \) in time, where \( l \) is the average size of generalized itemsets, \( m \) is the number of items, \( w(l) \) is the width of the lattice and \( d(l) \) 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]
and [17]. In the worst case, the number of GCFIs is equal to the number of GFI s, 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 ∪ 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 frequent, 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 $X_1$ 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 $\text{minsup} = 50\%$, 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.

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 $\text{SET-Main}$, the subordinate function called $\text{SET-Extend}$, recursively creates a subtree using the proposed method. The $\text{GenTaxChild}$ function produces a taxonomy-based child itemset while the $\text{GenJoinChild}$ 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 $\text{NewNode}$ function generates a new itemset under the current itemset. For the $\text{cSET}$ algorithm, line 11.s, 13.s and 17.s are ignored. Similar to SET, cSET uses $\text{SET-Main}$, $\text{SET-Extend}$, $\text{GenTaxChild}$ and $\text{GenJoinChild}$ as well as $\text{cSET-Property}$ and $\text{Hash}$ 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 $\text{NewNode}$ function in line 13.s and 17.s, we use the $\text{cSET-Property}$ 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 $\text{Hash}$ 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.
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 × 10 × 5 × 4 bytes ≈ 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.

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 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 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.
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 minsup 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 taxonomy 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.

**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.

<table>
<thead>
<tr>
<th>Minsup</th>
<th>Execution Time (sec)</th>
<th>#Frequent Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SET</td>
<td>cSET</td>
</tr>
<tr>
<td>Dataset: SynR250F5D1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>1.5</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>2.9</td>
<td>2.6</td>
</tr>
<tr>
<td>0.75</td>
<td>4.0</td>
<td>3.3</td>
</tr>
<tr>
<td>Dataset: MushroomR40F3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>90</td>
<td>0.06</td>
<td>0.02</td>
</tr>
<tr>
<td>80</td>
<td>0.31</td>
<td>0.02</td>
</tr>
<tr>
<td>70</td>
<td>1.09</td>
<td>0.05</td>
</tr>
<tr>
<td>60</td>
<td>2.92</td>
<td>0.05</td>
</tr>
<tr>
<td>50</td>
<td>9.48</td>
<td>0.13</td>
</tr>
<tr>
<td>Dataset: MushroomR24F5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0.14</td>
<td>0.01</td>
</tr>
<tr>
<td>90</td>
<td>1.41</td>
<td>0.03</td>
</tr>
<tr>
<td>80</td>
<td>4.59</td>
<td>0.06</td>
</tr>
<tr>
<td>70</td>
<td>12.88</td>
<td>0.13</td>
</tr>
<tr>
<td>60</td>
<td>29.94</td>
<td>0.28</td>
</tr>
<tr>
<td>50</td>
<td>76.67</td>
<td>0.59</td>
</tr>
<tr>
<td>Dataset: ChessR15F5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>2.36</td>
<td>0.01</td>
</tr>
<tr>
<td>98</td>
<td>15.41</td>
<td>0.01</td>
</tr>
<tr>
<td>96</td>
<td>25.45</td>
<td>0.02</td>
</tr>
<tr>
<td>94</td>
<td>65.69</td>
<td>0.02</td>
</tr>
<tr>
<td>92</td>
<td>110.75</td>
<td>0.05</td>
</tr>
<tr>
<td>90</td>
<td>179.53</td>
<td>0.06</td>
</tr>
</tbody>
</table>

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


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.

Thanarak Theeramunkong received 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.