Mining Nested Association Patterns

Ke Wang
Huiqing Liu
Department of Information Systems and Computer Science
National University of Singapore
wangk@iscs.nus.sg, liuhuiqi@iscs.nus.sg

Abstract

We introduce the framework of mining association patterns from nested databases. Two means to nest data items, namely, set and sequence, are considered. The term "collection" refers to a piece of data obtained by such nestings. A natural binary relation defines the generalization hierarchy among all collections. A *transaction database* is a set of given collections, called *transactions*. The problem of mining nested association patterns is to find all collections that are generalization of some minimum fraction of transactions, called *nested association patterns*. We sketch out the working idea of an algorithm for mining nested association patterns.

1 Introduction

The problem of mining association rules is to find all itemsets that are contained in some minimum fraction of transactions. Such itemsets are called association patterns. Following the work in [AIS93], several generalizations of association rules were considered. Multiple-level association rules [HF95, SA95] deal with items that are related by a concept hierarchy. Sequential patterns [AS95] allows a sequence of subtractions within a transaction ordered by the purchase time. Recently, [FMM96, SA96] have considered both quantitative and categorical data. While these works have substantially generalized the application domain of association rules, the data type allowed is still very simple, i.e., sets of atomic items. In many applications, items have descriptions and it is often the association between descriptions of items, rather than items themselves, that is interesting to the user. In addition, a description itself may be described by detailed descriptions which may be further described by other descriptions, and so on. Descriptions may or may not be ordered. In such cases, the context in which descriptions appear, i.e., the nesting relationship of descriptions, constitute the features of significance and should be captured in the patterns. The next example (also a running example) illustrates this point.

Example 1.1 *Consider a database of task definitions (or bills of materials, production schedules, libraries of programs, etc.). Each task is defined either by a set of subtasks* $\{a_1, \ldots, a_k\}$*, in which the execution order of subtasks does not matter, or by a sequence of subtasks* $< a_1, \ldots, a_k >$*, in which the sequence defines the execution order of subtasks. A subtask can be either atomic or recursively defined by other subtasks. Suppose that there are five tasks* $t_1, \ldots, t_5$ *in the database whose definitions are given by f function:*

$$f(t_1) = \{2, c\},\ f(t_2) = \{1, b, e\},\ f(t_3) = \{2, e\},$$
$$f(t_4) = \{1, b, d\},\ \ f(t_5) = \{2, b\},$$

where a, b, c, d, e *are subtasks defined as*

$$f(a) =< 1, 1 >,\ f(b) = \{1, 2\},\ f(c) =< 1, a, 2, a, 1 >,$$
$$f(d) = \{1, 2, 3, b\},\ \ f(e) =< c, 2 >$$

and $1, 2, 3$ *are atomic subtasks. We find the following subtask patterns in at least 60%, or three, tasks in the database* $\{t_1, \ldots, t_5\}$*: subtask pattern* $\{2\}$ *appears in tasks* t_1, t_3, t_5*, pattern* $\{b\}$ *appears in tasks* t_2, t_4, t_5*, and pattern* $\{<< 1, 1 >, 2 >\}$ *appears in* t_1, t_2, t_3*, where pair* $\{\ldots\}$ *means "appearing as a subset" and pair* $< \ldots >$ *means "appearing as a subsequence". For example, the last pattern says that for at least three tasks in* $\{t_1, \ldots, t_5\}$*, there is a subtask in which subtask 2 follows some subtask that executes subtask 1 sequentially twice.*

This example shows that association patterns augmented with nesting relationships can capture intertask dependencies and composition, which are essential for job scheduling, trigger systems, transaction and workflow processing, resource management, and consistency enforcement. In a broad range of applications, data are not necessarily represented in a flat form, nor do association patterns always occur in the form of common itemsets. Applications running on object-oriented databases are such examples, where an object is represented by a hierarchy of objects through type aggregation [GDV88]. Other examples are "bulk" data types (see [R92, SZLV93]) where data are typically divided into ordered or unordered groups, and members within a group may be further divided recursively. The interest

of "bulk" data types arises from applications that store nested relations, time-series data, meteorological and astronomical data streams, textual information, tree-structure file system, HTML document hierarchies on Web, DNA sequences, drug and chemical structures, and voices/sound/images/ video, etc. For instance, an HTML document tree is usually ordered and can easily go to 5 or more levels of nesting. While there have been many proposals on query languages and access support, few frameworks and algorithms exist for discovering interesting patterns from such semantically rich data.

In the rest of the paper, we introduce a framework of mining association patterns from nested data and discuss issues related to a mining algorithm, with an example included. It is not the objective of this short paper to present a formal mining algorithm.

2 The Problem

Definition 2.1 (The transaction database) *A* transaction database *is a quadruple (T, N, D, f), where T and N are sets of* terminal values *and* non-terminal values, *respectively. T and N are disjoint. D is a non-empty subset of N whose elements are called* transactions. *For each A in N, $f(A)$ is either a set $\{A_1, \ldots, A_p\}$ or a sequence $< A_1, \ldots, A_p >$, where A_i are values in $T \cup (N - D)$ and $p > 0$. For all transactions t in D, $f(t)$ has the same type, i.e., set or sequence. This type is called the* transaction type.

A_i are called *children* of A. The descendant relationship is defined in a natural way. We assume that no value is a descendant of itself A transaction database defines the universe of collections which forms the generalization space of transactions.

Definition 2.2 (Collections) *Every value in $T \cup N$ is a* collection *of itself. Assume that $x_1, \ldots, x_p$ are collections of values $A_1, \ldots, A_p$ in $T \cup (N - D)$. If $f(A) = \{A_1, \ldots, A_p\}$, $\{x_{i_1}, \ldots, x_{i_k}\}$ is a* collection *of A, where $1 \leq i_j \leq p$ and $k > 0$. If $f(A) =< A_1, \ldots, A_p >$, $< x_{i_1}, \ldots, x_{i_k} >$ is a* collection *of A, where $i_1 < \ldots < i_k$ and $k > 0$.*

Definition 2.3 (Full collections) *Every value in $T \cup N$ is the* full collection *of itself. Assume that $x_1, \ldots, x_p$ are full collections of $A_1, \ldots, A_p$ in $T \cup (N - D)$. If $f(A) = \{A_1, \ldots, A_p\}$, $\{x_1, \ldots, x_p\}$ is the* full collection *of A. If $f(A) =< A_1, \ldots, A_p >$, $< x_1, \ldots, x_p >$ is the* full collection *of A.*

That is, collections are obtained by partially expanding values A in N according to mapping $f(A)$, where "partially" means that some members in $f(A)$, but not all, may be dropped during expanding. Full collections are collections obtained by dropping no members. The following "weaker than" relationship defines a generalization hierarchy among collections, whose purpose is to compare the generalization power of patterns in the discovery process.

Definition 2.4 (Weaker than) *(a) Every value in $T \cup N$ is* weaker than *itself. (b) $\{x_1, \ldots, x_p\}$ is* weaker than *$\{x'_1, \ldots, x'_q\}$ If every x_i is a subset of some x'_{j_i}; $< x_1, \ldots, x_p >$ is* weaker than *$< x'_1, \ldots, x'_q >$ if every x_i is a subsequence of some x'_{j_i} such that $j_1 < \ldots < j_p$. (c) $\{x_1, \ldots, x_p\}$ or $< x_1, \ldots, x_p >$ is* weaker than *non-terminal value A if it is weaker than some collection of A. Two collections are* equivalent *if they are weaker than each other.*

Intuitively, x being weaker than x' means that x contains no more information than x' or that x is as general as x'.

Example 2.1 *Consider the transaction database (T, N, D, f) in Example 1.1, where $T = \{1, 2, 3\}$, $D = \{t_1, \ldots, t_5\}$, $N = D \cup \{a, b, c, d, e\}$, and f is as follows:*

$$f(t_1) = \{2, c\},\ f(t_2) = \{1, b, e\},\ f(t_3) = \{2, e\},$$
$$f(t_4) = \{1, b, d\},\ \ f(t_5) = \{2, b\},$$

$$f(a) =< 1, 1 >,\ f(b) = \{1, 2\},\ f(c) =< 1, a, 2, a, 1 >,$$
$$f(d) = \{1, 2, 3, b\},\ \ f(e) =< c, 2 >.$$

$< 1, 1 >$ is weaker than $< 1, < 1, 1 >, 2, < 1, 1 >, 1 >$, $<< 1, 1 >, 2 >$ is weaker than $<< 1, < 1, 1 >, 2, < 1, 1 >, 1 >, 2 >$, $<< 1, 1 >, 2 >$ is weaker than e. This implies that $\{<< 1, 1 >, 2 >\}$ is weaker than t_2 and that $\{<< 1, 1 >, 2 >\}$ is weaker than t_3.

Definition 2.5 (Nested association patterns) *Assume that (T, N, D, f) is a transaction database. Consider a collection x. The* support *of x is the number of transactions t in D such that x is weaker than t. For a user-specified minimum support minisup, x is* frequent *if the support of x is not less than minisup. x is* maximally frequent *if x is frequent and is not weaker than other frequent collections. x is a* nested association pattern *or simply* pattern *if x is maximally frequent. Given a transaction database (T, N, D, f), the problem of mining nested patterns is to find all patterns with respect to minisup.*

Testing whether a collection is maximally frequent requires to examine other collections, therefore, is difficult. We consider a necessary condition of being maximally frequent. This condition can be used to prune some non-maximally frequent collections.

Definition 2.6 (Non-redundant collections) *(a) Every value in $T \cup N$ is non-redundant. (b) A collection $< x_1, \ldots, x_k >$ is non-redundant if and only if every x_i is non-redundant. (c) A collection $\{x_1, \ldots, x_k\}$ is non-redundant if and only if every x_i is non-redundant and no x_i is weaker than x_j for $i \neq j$.*

For example, $\{\{a,b\}\}$ and $\{<< 1 >, 2 >\}$ are non-redundant, but $\{\{a\},\{a,b\}\}$ and $\{<< 1 >, 2 >, < 2 >\}$ are redundant.

Theorem 2.1 *Every maximally frequent collection is non-redundant, but not the converse.*

In fact, if a frequent collection x is redundant, it must be equivalent to some non-redundant collection (which is also frequent), and by the above definition x is not maximally frequent. $\{\{1\}\}$ is non-redundant, but is not maximally frequent because it is weaker than frequent non-redundant collection $\{b\}$. In Example 2.2 below, $\{\{1\}\}$ is non-redundant, but is not maximally frequent because it is weaker than frequent non-redundant collection $\{b\}$.

Example 2.2 *Consider the transaction database (T, N, D, f) in Example 2.1. Collection $\{2\}$ is supported by transactions t_1, t_3, t_5, $\{b\}$ is supported by transactions t_2, t_4, t_5, and $\{<< 1, 1 >, 2 >\}$ is supported by transactions t_2 and t_3. It can be verified that $\{<< 1, 1 >, 2 >\}$ is also supported by t_1. Therefore, if minisup is 60%, $\{2\}, \{b\}, \{<< 1, 1 >, 2 >\}$ are frequent. Clearly, $\{2\}, \{b\}, \{<< 1, 1 >, 2 >\}$ are non-redundant. Finally, $\{2\}, \{b\}, \{<< 1, 1 >, 2 >\}$ are maximally frequent because each is not weaker than a frequent non-redundant collection other than itself. All other frequent non-redundant collections are weaker than some of these collections. For example, $\{<< 1 >, 2 >\}$ is weaker than $\{<< 1, 1 >, 2 >\}$, $\{\{1,2\}\}$ is weaker than $\{b\}$, etc. Therefore, $\{2\}, \{b\}, \{<< 1, 1 >, 2 >\}$ are all and only patterns.*

3 The Issues

We examine issues related to mining nested association patterns.

3.1 Representing the transaction database

First, we need to represent the transaction database (T, N, D, f). The straightforward method is to store the full collection for each transaction in D, in which the full collection of a non-terminal value A will be repeatedly stored in every transaction having A as a descendant. This will considerably increase the database size and cause some update anomalies when the nesting function $f(A)$ is updated. More seriously, since non-terminal values are replaced by their full collections, no patterns containing non-terminal values will be discovered.

One assumption made in most data mining problems is that the number of distinct values is small compared to the number of tuples in the database. In our mining problem, we assume that the number of values in $T \cup (N - D)$ (i.e., non-transaction values) is smaller than the number of transactions in D (i.e., the database size). We will store the definition $f(A)$ for values $A \in T \cup (N - D)$ in the memory and store the definition $f(t)$ for transactions $t \in D$ on the disk. The mapping $f(A)$ for values $A \in N - D$ are represented by a directed acyclic graph G defined as follows.

1. For each value $A \in T \cup (N - D)$, we create $k + 1$ nodes $A^0, A^1, \ldots, A^k$ in G, called *copies* of A, where k is the maximal number of occurrences of A in a single sequence $f(B)$ for $B \in N - D$. If $f(A)$ is a sequence, nodes A^i are marked by box $\Box$. Intuitively, A^0 represents A in a set and A^i, $i \geq 1$, represents the ith occurrence of A in a sequence. For each value A in $N - D$, if $f(A) =< B_1, \ldots, B_n >$, there is an arc from every copy of A to the jth copy of B_i if B_i appears as the jth occurrence of itself.

2. For each value A in $N - D$, if $f(A) = \{B_1, \ldots, B_n\}$, there is an arc from every copy of A to $B_1^0, \ldots, B_n^0$. Note that G does not contain mappings $f(t)$ for transactions t; it contains only the definition for values in $N - D$. G is stored in the memory.

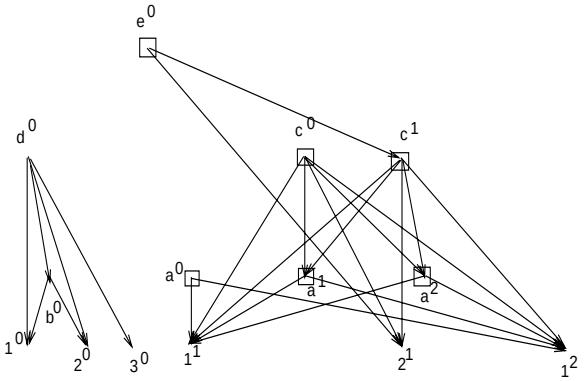

Figure 1: Graph G

Figure 1 shows the graph G for values in $N - D$ for the transaction database in Example 2.1.

3.2 Representing collections

Second, we need to represent collections that may contain both sets and sequences. A *k-collection* refers to a collection in which there are k occurrences of values in $T \cup (N - D)$. Each value occurrence in a k-collection can be represented by a path in graph G, called a *G-path*. The algorithm makes multiple passes over the transaction file D, computing frequent non-redundant k-collections for increasing k. In the first pass, we find all frequent non-redundant 1-collections, denoted F_1, each being represented by a G-path. In the kth pass, $k > 1$, we compute all frequent non-redundant k-collections, denoted F_k, from F_{k-1}.

Each k-collection can be represented by a sequence $p_1 \ldots p_k$, where p_i is the G-path for the ith value occurrence in the collection, from left to right. A *nesting*

tree of k leaves can be "assembled" by the k G-paths $p_1, \ldots, p_k$ in a natural way, that is, by treating G-paths p_i as paths of the tree ordered as in $p_1 \ldots p_k$. We represent a k-collection interchangeably by either a sequence of k G-paths or the corresponding nesting tree.

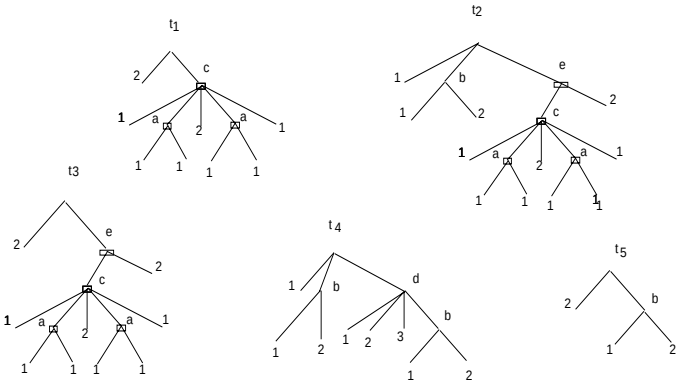

Figure 2: Nesting structures of transactions

Example 3.1 *Continue with the task database* (T, N, D, f) *in Example 2.1. Assume that minisup is 60%, i.e., a frequent collection must be supported by at least three transactions. The graph G is as in Figure 1. For convenience, the nesting structures of transactions* $t_1, \ldots, t_5$ *are displayed in Figure 2.* F_1 *is produced in Table 1. In particular, there are 14 G-paths representing 6 frequent 1-collections (in the third column). The second column gives the supporting transactions of these G-paths.*

3.3 Constructing collections

We need a rule to construct larger frequent collections from smaller ones. Relational join does not work here because objects are not simple tuples, but complex collections. However, the idea of constructing a larger frequent collections from two smaller frequent collections still applies. The following theorem forms the basis of computing F_k from F_{k-1}.

Theorem 3.1 *Let* p_i *be a G-path. Every frequent and non-redundant k-collection* $p_1 \ldots p_{k-1} p_k$ *is constructed by two frequent and non-redundant* $(k-1)$*-collections* $p_1 \ldots p_{k-2} p_{k-1}$ *and* $p_1 \ldots p_{k-2} p_k$ *such that* p_{k-1} *is not a prefix of* p_k *and vice versa.*

In Theorem 3.1, we say that $p_1 \ldots p_{k-2} p_{k-1}$ is *extended* by $p_1 \ldots p_{k-2} p_k$. The construction in Theorem 3.1 gives a superset of F_k, called *k-candidates*. The actual frequent k-collections in F_k are found by counting the support of k-candidates during a pass over the transaction file D. It does not work to simply treat G-paths as items and find nested patterns as mining large itemsets in [AIS93, AS94] because the connectivity among G-paths and the nesting hierarchy of the data are important.

Example 3.2 *Continue with Example 3.1. Table 2 shows* F_2 *and* F_3*. In computing* F_2*, 11* Π*-paths representing frequent and non-redundant 2-collections are generated, as shown in Table 2(a).* p_7, p_8, p_9, p_{14} *are not extended for reasons to be given in Example 3.3 shortly. It can be verified that all other 2-collections not shown in Table 2(a) are either redundant or not frequent. For example,* $p_{11}p_{12} = \{\{2\}, < 2 >\}$ *and* $p_{11}p_{13} = \{\{2\}, << 1 >>\}$ *are not frequent because they are only supported by* t_2*. In computing* F_3*, 3 frequent and non-redundant 3-collections are generated, shown in Table 2(b). All other 3-collections not shown are either redundant or not frequent. In computing* F_4*,* $p_6p_7p_{12}$ *and* $p_6p_7p_5$ *produces* $p_6p_7p_{12}p_5$*, representing redundant collection* $\{<< 1, 1 >, 2 >, < 2 >\}$*. Thus,* F_4 *is empty and no larger frequent collections will be generated.*

3.4 Pruning strategies

In mining nested association patterns, the search space is much larger than the case of flat transactions in [AIS93, AS94] due to the large number of ways of nesting values. A effective pruning is crucial to the performance of the algorithm. However, the pruning of non-maximally frequent collections is much less straightforward than that in mining association rules from flat transactions. As we shall see later, in general, a non-maximally frequent collection can be used to produce a maximally frequent collection, therefore, pruning all non-maximally collections after each iteration could affect the completeness of the answer. We will discuss two pruning strategies that effectively cut down the search space and ensure the completeness of the answer.

It is possible that nesting trees produced by different pairs of $(k-1)$-collections in Theorem 3.1 are *isomorphic* in the sense that they differ only in superscripts i in nodes of form A^i. We can show that only one of such nesting trees, called a *natural* nesting tree, needs to be considered. Intuitively, a nesting tree is *natural* if superscript i for a node of the form A^i corresponds to the order of the occurrence of A under its parent node. Extending a non-natural nesting tree $p_1 \ldots p_{k-1}$ always produces a non-natural nesting tree $p_1 \ldots p_{k-1} p_k$ because the prefix $p_1 \ldots p_{k-1}$ is non-natural. Thus, our strategy is

Strategy I. Do not extend a non-natural nesting tree in Theorem 3.1.

Example 3.3 *In Table 1,* p_7, p_8, p_9 *and* p_{14} *are non-natural and will not be extended in later passes according to Strategy I. In Table 2,* p_5p_{14} *is non-natural. Interestingly,* p_5p_{14} *extends* p_5p_{13} *into natural* $p_5p_{13}p_{14}$*. Therefore, a non-natural nesting tree may be useful to extend a natural nesting tree into a natural one.*

G-paths	supporting transactions	collections	non-natural
$p_1 : (2^0)$	t_1, t_3, t_5	$\{2\}$	
$p_2 : (b^0)$	t_2, t_4, t_5	$\{b\}$	
$p_3 : (b^0, 1^0)$	t_2, t_4, t_5	$\{\{1\}\}$	
$p_4 : (b^0, 2^0)$	t_2, t_4, t_5	$\{\{2\}\}$	
$p_5 : (c^0, 2^1)$	t_1, t_2, t_3	$\{< 2 >\}$	
$p_6 : (c^0, a^1, 1^1)$	t_1, t_2, t_3	$\{<< 1 >>\}$	
$p_7 : (c^0, a^1, 1^2)$	t_1, t_2, t_3	$\{<< 1 >>\}$	yes
$p_8 : (c^0, a^2, 1^1)$	t_1, t_2, t_3	$\{<< 1 >>\}$	yes
$p_9 : (c^0, a^2, 1^2)$	t_1, t_2, t_3	$\{<< 1 >>\}$	yes
$p_{10} : (d^0, 1^0)$	t_2, t_4, t_5	$\{\{1\}\}$	
$p_{11} : (d^0, 2^0)$	t_2, t_4, t_5	$\{\{2\}\}$	
$p_{12} : (e^0, 2^1)$	t_1, t_2, t_3	$\{< 2 >\}$	
$p_{13} : (e^0, c^1, 1^1)$	t_1, t_2, t_3	$\{<< 1 >>\}$	
$p_{14} : (e^0, c^1, 1^2)$	t_1, t_2, t_3	$\{<< 1 >>\}$	yes

Table 1: Computing F_1

The second strategy is to prune non-maximally frequent collections that will not be used in a later iteration. Unlike the non-redundancy, however, k-collection $p_1 \ldots p_k$ being maximally frequent implies that $(k-1)$-collections $p_1 \ldots p_{i-1} p_{i+1} \ldots p_k$ are not maximally frequent because $p_1 \ldots p_{i-1} p_{i+1} \ldots p_k$ are weaker than $p_1 \ldots p_k$. Therefore, non-maximally frequent collections are needed to construct maximally frequent collections and cannot be pruned until all frequent non-redundant collections are constructed. However, there is a special case where pruning non-maximally frequent collections in F_i does not affect computing F_j for $j > i$: if extending l by l' generates a frequent non-redundant collection, l is not maximally frequent anymore because it is weaker than the newly constructed collection, and l will not be used in a later pass because it contains too few values. This gives

Strategy II. If extending l by l' generates a frequent non-redundant collection, prune l.

Example 3.4 *We prune all non-maximally frequent collections in $F_1 \cup F_2 \cup F_3$. By Strategy II, there is no need to consider collections in F_k that were extended into some collections in F_j, where $j > k$. After the pruning, only three collections remain, namely, $\{2\}$, $\{b\}$, and $\{<< 1, 1 >, 2 >\}$, which are the patterns returned.*

4 Related Work

Since [AIS93], several generalization of association rules have been proposed. Quantitative association [SA96, FMM96] considers real values and is quite different from our work. Sequential patterns [AS95] are special nested association patterns where a transaction $f(t)$ is a sequence $< t_1, \ldots, t_n >$ of times and mappings $f(t_i)$ give itemsets purchased at time t_i. The computation of support in the case of a nesting hierarchy is very different from that in the case of an *isa*-hierarchy for multi-level association rules [HF95, SA95]. For example, a transaction supports value $StandardEngine$ defined by $f(StandardEngine) = \{1000cc, FourCylinder\}$ if it contains value $StandardEngine$, implying that the engine is both $1000cc$ and $FourCylinder$. On the other hand, a transaction supports value $Milk$ having children $LowFat$ and $HighFat$ in an *isa*-hierarchy if it contains either $LowFat$ or $HighFat$, not necessarily both. In addition, a nesting hierarchy generalizes data by subsetting and subsequencing while preserving nesting relationships, whereas an *isa*-hierarchy generalizes data by substituting low level concepts with high level ones. As a result, nested association patterns contain nesting relationships, whereas multi-level association patterns are flat sets of concepts (or items). Nevertheless, one deals with data enrichment by an inheritance hierarchy and the other deals with data enrichment by an aggregation hierarchy.

5 Conclusion

As the amount of data available on-line grows rapidly, we found that more and more data are organized into a hierarchy without a fixed schema. Mining interesting association that appears deep down in the hierarchy and preserves the hierarchical abstraction is a powerful tool for analyzing data dealt with by many modern applications, such as job scheduling, trigger systems, consistency enforcement in transaction and workflow environments, resource management, exploit of repository technology [BD94] and reuse engineering, and analysis of complex objects such as Web documents

sequences of G-paths	supporting transactions	collections	non-natural
p_3p_4	t_2, t_4, t_5	$\{\{1,2\}\}$	
p_3p_{11}	t_2, t_4, t_5	$\{\{1\},\{2\}\}$	
p_4p_{10}	t_2, t_4, t_5	$\{\{2\},\{1\}\}$	
p_5p_{13}	t_1, t_2, t_3	$\{< 2 >, << 1 >>\}$	
p_5p_{14}	t_1, t_2, t_3	$\{< 2 >, << 1 >>\}$	yes
p_6p_7	t_1, t_2, t_3	$\{<< 1, 1 >>\}$	
p_6p_{12}	t_1, t_2, t_3	$\{<< 1 >>, < 2 >\}$	
p_6p_5	t_1, t_2, t_3	$\{<< 1 >, 2 >\}$	
$p_{10}p_{11}$	t_2, t_4, t_5	$\{\{1,2\}\}$	
$p_{13}p_{14}$	t_1, t_2, t_3	$\{<< 1, 1 >>\}$	
$p_{13}p_{12}$	t_1, t_2, t_3	$\{<< 1 >, 2 >\}$	

(a) F_2

sequences of G-paths	supporting transactions	collections	non-natural
$p_5p_{13}p_{14}$	t_1, t_2, t_3	$\{< 2 >, << 1, 1 >>\}$	
$p_6p_7p_{12}$	t_1, t_2, t_3	$\{<< 1, 1 >>, < 2 >\}$	
$p_6p_7p_5$	t_1, t_2, t_3	$\{<< 1, 1 >, 2 >\}$	
$p_{13}p_{14}p_{12}$	t_1, t_2, t_3	$\{<< 1, 1 >, 2 >\}$	

(b) F_3

Table 2: F_2 and F_3

or parts in CAM/CAD. The approach of flattening the database and applying conventional mining tools does not work because the database size may increase considerably, update becomes very difficult, and the ordering of values at different levels is lost. Mining nested association patterns requires a new approach. This paper presented initial steps of such approachs.

References

[AIS93] R. Agrawal, T. Imielinski, A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD 1993*, pages 207-216

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB 1994*, pages 487-499

[AS95] R. Agrawal and R. Srikant. *Mining sequential patterns*. In *ICDE 1995*, pages 3-14

[BD94] P.A. Bernstein and U. Dayal, "An overview of repository technology", VLDB 94, 705-713

[FMM96] T. Fukuda, Y. Morimoto, S. Morishita. Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization. In *SIGMOD 1996*, pages 13-23

[GDV88] M.J. Garey, D.J. DeWitt, S.L. Vandengerg. A data model and query language for EXODUS. In *ACM SIGMOD 1988*

[HF95] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *VLDB 1995*, pages 420-431

[R92] J. Richardson. Supporting lists in a data model. In *VLDB 1992*, pages 127-138

[SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB 1995*, pages 407-419

[SA96] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. *SIGMOD 1996*, pages 1-12

[SZLV93] B. Subramanian, S.B. Zdonik, T. W. Leung, and S. L. Vandenberg. Ordered types in the AQUA data model. In *Proceedings of 4th International Workshop on Database Programming Languages*, 1993, pages 115-135