MINING IS-PART-OF ASSOCIATION PATTERNS FROM SEMISTRUCTURED DATA

KE WANG
School of Computing Science
Simon Fraser University
E-mail: wangk@cs.sfu.ca

HUIQING LIU
BioInformatics Centre
National University of Singapore
Email: huiqing@bic.nus.edu.sg

One example of semistructured data sources is the World Wide Web (WWW). In the semistructured world, the individual schema contained in each object has replaced the external schema of the data. An immediate implication on data mining is that it has to deal with both data and schemas. This requires the data generalization to trace the role of objects and handle structural irregularity, arbitrary nesting of ordered and unordered types, and cyclic object references. We introduce the framework of *is-part-of association patterns* to address the issue. We show applications of mining is-part-of association patterns in several disparate domains.

1 Introduction

1.1 Motivation

As the amount of data available on-line grows rapidly, we find that more and more of the data are semistructured. Semistructured data arise when the source does not impose a rigid structure such as the Web and when data are combined from several heterogeneous sources such as a data warehouse. See, for example, [1,6,15] for motivation and the workshop [20] for some recent works. Semistructured data have two essential features:

Structure. Semistructured data do have structures. Figure 1 shows a segment of semistructured movie objects maintained by IMDb (http://us.imdb.com). Each circle plus the text inside it represents an object, e.g., a HTML subdocument, and its identifier, e.g., the URL. The arrows and their labels represent object references and their roles. Object references can be cyclic, ordered, and up to an arbitrary level. Object references and their labels, often identifiable by special tags or a grammar, are important structural information used by semistructured query languages (e.g., [6,15]). A recent review has revealed that nearly always, references to important objects

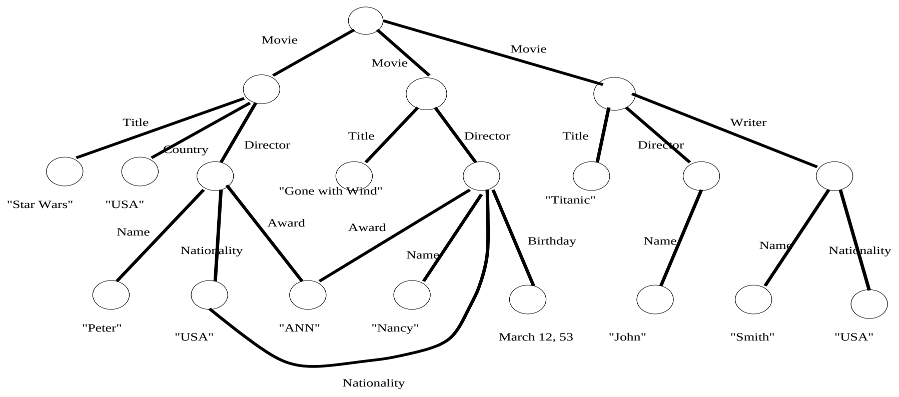

Figure 1. A segment of the internet movie database

are tagged rather than in the form of free-running text [8].

Irregularity of structure. Semistructured data have no fixed schema or object class; instead, every object contains its own schema, i.e., self-describing. This semistructured nature provides the flexibility needed in a heterogeneous, dynamic environment [15]: the director/actor information may be found in several heterogeneous sources on the Web, movies in different categories or actors playing different roles may have different structures, standards and features of movies may be added and deleted, etc. As a consequence, the user is not expected to know all the details of the schema, and queries over the schema are as important as standard queries over the data.

These features have important implications on a data mining task. The structural feature implies that the mining task can focus on the labeled object references but needs to handle structural features like ordering, nesting, and cyclicity. The structural irregularity is more influential: the mining task has to deal with both objects and the meaning or role of objects.

Other sources of semistructured data are SGML, BibTex or LaTeX files, electronic shopping transactions, scientific databases, libraries of programs, production schedules, and more generally, digital libraries. In such a broad range of applications, partial orderings among objects are common and typical orderings are important knowledge about the data. This is illustrated in the following running example.

Example 1.1 Five transactions are recorded in an electronic shopping application: &$t1$, &$t2$, &$t3$, &$t4$, &$t5$, as defined in Figure 2. &a, &c, &e are identifiers for subtransactions, and $Photo$, $Pizza$, $Paint$, $Floor$, $Wall$, $Furniture$

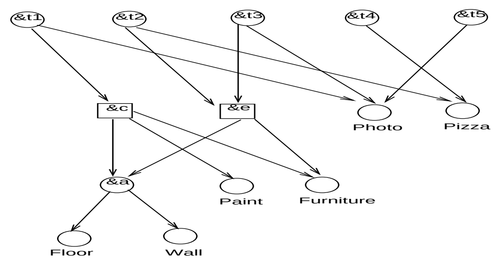

Figure 2. The OEM graph for an electronic shopping database

are items or services requested. A circled node unorders its subnodes, whereas a squared node orders its subnodes in the order shown. For example, $\&t1$ requests that services $\&a$, $Paint$, $Furniture$ be done sequentially and that $Photo$ taking be done either before or after all these, where identifier $\&a$ represents two unordered services $Floor$ and $Wall$. Constraints like these are typically captured by on-line form-filling. Interestingly, transactions $\&t1$ and $\&t3$ share the following similarity: $\&a$ precedes $Furniture$, and $Photo$ either precedes or follows both, written as $\{< \&a, Furniture >, Photo\}$. Finding this similarity requires ignoring the difference of identifiers $\&c$ and $\&e$ at intermediate levels. In the real world, each service can be further described by information like category, special offer, type of payment, and site/provider information, and similarities could exist at such levels.

1.2 New requirements for generalizing data

Semistructured data, formally defined in Section 2, are really an *is-part-of* hierarchy (in fact, a graph) in which subobjects (at a lower level) are components of a superobject (at a higher level). Very often, objects share similarities at low levels, though not at high levels. This is in contrast with the *is-a* hierarchy (i.e., a concept hierarchy) in [11,16] where similarities tend to be shared at higher levels for more general concepts. A more important difference is that generalizing data in an *is-part-of* hierarchy needs to trace the roles or labels followed. For semistructured data, tracing the roles of objects is necessary because no external schema is provided for this information. Generalization in an *is-part-of* hierarchy also needs to deal with arbitrary nesting of ordered and unordered types, and cyclic object references. Finally, an *is-part-of* hierarchy is likely to be very large because it is the actual data, not knowledge of the data.

1.3 Related work

Since [3], most research on association rules has focused on improving the speed of the algorithm, see [12] for a list, and several have enlarged the application domain [5,10,11,13,16,17]. All of these dealt with either flat files or tables over fixed schemas. Using an *is-a* hierarchy has led to interesting results [9,11,16,18], but offered little for an *is-part-of* hierarchy as desired for semistructured data. Research on semistructured data, on the other hand, mainly dealt with modeling, searching, structure extracting, and information exchanging [1,15,20]. [19] classified semistructured documents into predefined types using the vector space model. We considered association, not classification, and treated structural properties such as nesting/ordering/referencing, which are ignored by any vector space model, as the first-class citizen. This also distinguishes our approach from information retrieval where a document is treated as a set of keywords.

We study the problem of mining association patterns from semistructured data, called *is-part-of association patterns*. We formally define the problem in Section 2. Section 3 highlights applications of the problem. Section 4 presents an algorithm. Section 5 concludes the paper.

2 The Problem

2.1 Data representation

We adopt the *Object Exchange Model (OEM)* for semistructured data used in [2,1,6,15]. In OEM, (a) every object o is identified by an *identifier* $\&o$; (b) the role of an object is described by a *label*; (c) the *value* of an object is either an *atomic* quantity, or a *bag* of subobjects $\{l_1 : \&o_1, \ldots, l_k : \&o_k\}$, or a *list* of subobjects $< l_1 : \&o_1, \ldots, l_k : \&o_k >$, where $\&o_i$ is an object identifier and l_i is the label of $\&o_i$. As usual, the order of subobjects in a bag does not matter, but it does in a list, and repeating of subobjects is allowed in a bag or a list. $val(\&o)$ denotes the value of the object with identifier $\&o$. A main feature of OEM is that it is self-describing: each object contains its own schema $l_1, \ldots, l_k$ for subobjects. Table 1 shows the OEM in Example 1.1 in which $Floor, Wall, Paint, Furniture, Photo, Pizza$ are atomic quantities. Labels not used in that example.

OEM can be represented by a labeled, directed, and possibly cyclic multigraph. Each node represents an object identifier. There is an edge $(\&o, \&o_i)$ labeled l_i if subobject $l_i : \&o_i$ belongs to $val(\&o)$. For a bag $val(\&o)$, the node for $\&o$ is circled and subnodes of $\&o_i$ are unordered. For a list $val(\&o)$, the node for $\&o$ is squared and subnodes of $\&o_i$ are ordered. Figure 2 shows

Table 1. The OEM for an electronic shopping database

transactions	subtransactions
$val(\&t1) = \{Photo, \&c\}$	$val(\&a) = \{Floor, Wall\}$
$val(\&t2) = \{Pizza, \&e\}$	$val(\&c) =< \&a, Paint, Furniture >$
$val(\&t3) = \{Photo, \&e\}$	$val(\&e) =< \&a, Furniture, Furniture >$
$val(\&t4) = \{Pizza\}$	
$val(\&t5) = \{Photo\}$	

the OEM graph for the data in Example 1.1.

2.2 Data generalization

Informally speaking, a generalization is a piece of partial information about an object, and an association pattern is a generalization frequently shared by objects. We can define a generalization of an object as a result of dropping subobjects and recursively generalizing the remaining subobjects. However, for the sake of integration with the algorithm in Section 4, we take a "constructive" approach that defines a generalization in terms of "smallest" generalizations. The idea is to "glue" reference paths of several descendant objects into a reference tree for representing a generalization. We formalize this idea below.

Reference-paths. Consider a *simple* node path $\&o, v_1, \ldots, v_n$ (i.e., in which only the last node v_n can repeat) in the OEM graph G. A *reference-path* starting at $\&o$ has the form $[\&o, l_1, v_1^{j_1}, \ldots, l_n, v_n^{j_n}]$. Each l_i is a label on edge (v_{i-1}, v_i) (treating $\&o$ as v_0), and each superscript j_i specifies an occurrence of subobject $l_i : v_i$ in $val(v_{i-1})$. If v_n repeats v_{n-i}, called the *ith ancestor* of v_n, $v_n^{j_n}$ is replaced with $Ans_i^{j_n}$. The special identifier Ans_i is the alias of the ith ancestor v_{n-i}, and we will explain the reason for introducing Ans_i in Section 2.3. A reference-path starting at $\&o$ contains all the information about how an occurrence of descendant v_n is nested within object $\&o$, and the sequence of labels $l_1, \ldots, l_n$ explains the role of v_n as a descendant of object $\&o$.

Reference-trees. Several reference-paths define a *reference-tree*. For reference-paths $p_1, \ldots, p_k$ starting at $\&o$, where no p_i is a prefix of another p_j, the reference-tree defined by sequence $p_1, \ldots, p_k$ is the "prefix tree" of these reference-paths obtained by "gluing" together common prefixes as much as possible such that p_i is the ith root-to-leaf path in the left-to-right order. $p_1 \ldots p_k$ denotes the resulting reference-tree. Consider the following reference-paths in Figure 2:

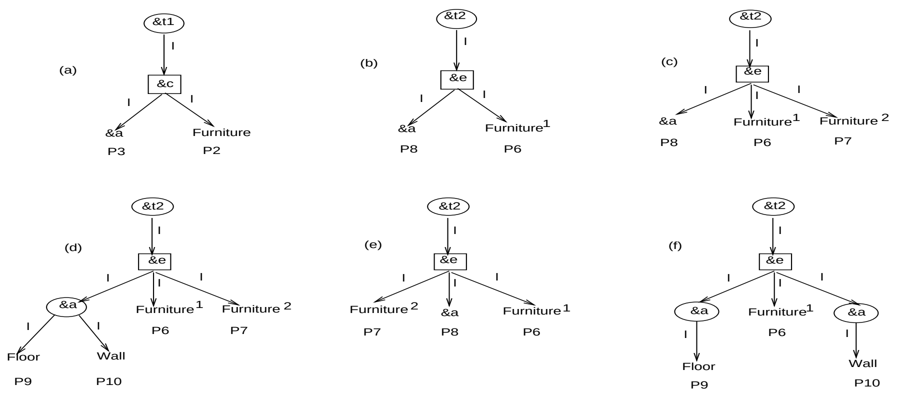

Figure 3. Reference-trees

$p_2 = [\&t1, l, \&c, l, Furniture]$ $\quad$ $p_7 = [\&t2, l, \&e, l, Furniture^2]$
$p_3 = [\&t1, l, \&c, l, \&a]$ $\quad$ $p_8 = [\&t2, l, \&e, l, \&a]$
$p_6 = [\&t2, l, \&e, l, Furniture^1]$ $\quad$ $p_9 = [\&t2, l, \&e, l, \&a, l, Floor]$
$p_{10} = [\&t2, l, \&e, l, \&a, l, Wall]$,

where l is an universal label. The superscripts of $Furniture$ in p_6 and p_7 denote the two occurrences of subobject $l : Furniture$ in $val(\&e)$. All occurrence numbers for non-repeating subobjects are omitted. Figure 3 shows six reference-trees p_3p_2, p_8p_6, $p_8p_6p_7$, $p_9p_{10}p_6p_7$, $p_7p_8p_6$, and $p_9p_6p_{10}$.

Generalizations. As far as generalizing an object is concerned, identifiers at non-leaf nodes in a reference-tree are not important. The term *generalization* refers to a reference-tree with all identifiers at non-leaf nodes ignored. The generalization represented by a reference-tree with subtrees $s_1, \ldots, s_k$ labeled $l_1, \ldots, l_k$ is written in the nested form $\{l_1 : g_1, \ldots, l_k : g_k\}$ for unordered subtrees s_i's, or $< l_1 : g_1, \ldots, l_k : g_k >$ for ordered subtrees s_i's, where g_i's are (recursively) generalizations for subtrees s_i's. For example, in Figure 3, reference-tree $p_9p_{10}p_6p_7$ represents generalization $\{< Floor, Wall\}, Furniture, Furniture >\}$. Note that any object identifier $\&o$ and its value $val(\&o)$ are generalizations.

2.3 The interestingness measure

Weaker than. Some generalizations are more "informative" than others. This is compared by the"weaker than" relation below. (i–basis) An identifier (including Ans_i) is *weaker than* itself. (ii–subbag) Generalization

$\{l_1 : g_1, \ldots, l_p : g_p\}$ is *weaker than* generalization $\{l'_1 : g'_1, \ldots, l'_q : g'_q\}$ if each g_i is weaker than some g'_{j_i} such that $l_i = l'_{j_i}$ and j_i is distinct for distinct i. (iii–sublist) Generalization $< l_1 : g_1, \ldots, l_p : g_p >$ is *weaker than* generalization $< l'_1 : g'_1, \ldots, l'_q : g'_q >$ if each g_i is weaker than some g'_{j_i} such that $l_i = l'_{j_i}$ and $j_1 < \ldots < j_p$. (iv–expansion) Generalization $\{l_1 : g_1, \ldots, l_p : g_p\}$ or $< l_1 : g_1, \ldots, l_p : g_p >$ is *weaker than* an identifier $\&o$ if it is weaker than $val(\&o)$.

In words, g is weaker than g' if all information in g can be found in g'. Therefore, an object $\&o$ has generalization g if and only if g is weaker than $\&o$. For example, $\{< \{Floor, Wall\}, Furniture, Furniture >\}$, $\{< \&a, Furniture >\}$, $\{< \{Floor\}, Furniture, Furniture >\}$ all are weaker than $\{< \&a, Furniture, Furniture >\}$, which is weaker than $\&t2$ and $\&t3$. Indeed, $\&t2$ and $\&t3$ share all the above generalizations.

Handling cyclic generalizations. We now can explain the "magic" of using special identifiers Ans_i in a cyclic reference-path. Consider the two cyclic generalizations (a) and (b) in Figure 4. Intuitively, (a) is "weaker than" (b) because all information in the former are found in the latter (identifiers at non-leaf nodes are not important). Thanks to the alias Ans_3, this is indeed recognized in their tree representations (a') and (b'). If the original identifiers $\&x$ and $\&y$ were used instead of Ans_3 in (a') and (b'), the "weaker than" relationship would not be found due to condition (i) of "weaker than". Replacing an identifier $\&o$ in a generalization g with any other generalization

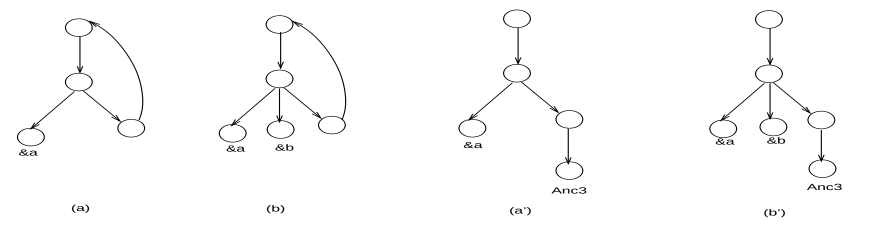

Figure 4. (a,b) using Ans_i, (a',b') not using Ans_i

weaker than $\&o$ always yields a generalization weaker than g. By considering only generalizations that are not weaker than any other generalizations, we can focus on generalizations containing identifiers at highest possible levels. This motivates the mining problem below.

Definition 2.1 (Mining is-part-of association patterns) Consider the input given by an OEM graph G, a minimum support MINISUP, and a collection of nodes in G called *transactions.* The *support* of a generalization g is the number of transactions $\&t$ such that g is weaker than $\&t$. g is *frequent* if the support of g is not less than MINISUP. g is *maximally frequent* if g is frequent and is not weaker than any other frequent generalization. g is a *is-part-of association pattern* or simply *pattern* if g is maximally frequent. The mining problem is to find all patterns.

Example 2.1 Consider the OEM graph in Figure 2 with $\&t1, \ldots, \&t5$ being transactions. Generalization $\{< \&a, Furniture >\}$ has support 3 because it is weaker than $\&t1, \&t2, \&t3$. $\{Photo\}$ also has support 3. Therefore, if MINISUP is 60%, i.e., 3, $\{< \&a, Furniture >\}$ and $\{Photo\}$ are frequent. Of course, any other generalization weaker than $\{< \&a, Furniture >\}$ are also frequent but not maximally frequent. One can verify that $\{< \&a, Furniture >\}$ and $\{Photo\}$ are the only maximally frequent generalizations, that is, patterns.

3 Applications

The following list gives a taste of the application of the proposed mining problem.

Electronic shopping. Example 1.1 shows one way of discovering electronic shopping patterns, i.e., by modeling partially ordered services as semistructured data and discovering typical partial orderings. The manager can use such patterns to organize service chains more effectively. We can also discover customers' interests and access patterns, as described below.

Interests/access patterns on WWW. Detecting user's interests and browsing patterns on the Web can help organize Web pages and attract more businesses. This can be modeled as mining is-part-of association patterns from a collection of hyperlinked Web pages that were accessed. Each page is identified by its URL and represented by a node in the OEM graph, and each transaction corresponds to the entry page of a browsing session. By labeling pages with either topics or site information, is-part-of association patterns capture either user's interests or access patterns.

Event and causality analysis. Example 1.1 is a special case of a broad range of applications such as job scheduling, dependency discovery, workflow and process management, resource management and discovery, and medical diagnosis. In such applications, an object represents either an atomic or complex event (e.g., a transaction, a production schedule, a bill-of-materials, a history of symptoms and diseases, etc), and an is-part-of association pattern is a regularity about how events are typically composed and how subevents are dependent on each other.

Schema discovery. Very often, most schemas, though not all, are typical to objects representing similar concepts. Discovering typical schemas can help specify queries and guide browsing [14,20]. Another motivation for discovering typical schemas is to build a structured layer above a less structured one, thus provide the benefits of standard methods [1]. Within our framework, each is-part-of association pattern is a typical schema if identifiers at leaf nodes are considered matching any identifiers in the data.

Classification of chemicals/proteins/living things. Chemical information systems organize chemical compound files into semistructured trees in which further information about each fragment occurs at each successively lower level. Classification of chemical structures is based on typical fragments in such trees. Protein structure classification also depends critically on identifying structural similarity (see CATH at http://www.biochem.ucl.ac.uk/bsm/cath/, for example). In biology, determining phylogenetic relationships is based on nested sets of derived characters (apomorphies) shared by lineages [7]. Mining is-part-of association patterns is the core of these tasks.

Text clustering and searching. The organizational and topical structures within a text document are usually ignored in information retrieval. In the semistructured view of documents, a subdocument is labeled by its topic, and keywords are atomic objects. An is-part-of association pattern for such documents contains frequently co-occurred keywords grouped according to topics. Using such topic-based associations, the search for IBM catalog prices for personal computers will not return things such as an advertisement for an "I Bought Mac" T-shirt and the VLDB96 home page. This approach is attractive in that the topical structure is contained in the data itself, not from the background knowledge.

4 The Algorithm

In this section, we describe an algorithm for mining is-part-of association patterns. We do not assume that the OEM graph G fits in memory. An important property of our algorithm is to traverse simple paths of G in the depth-first manner. Since several supernodes may reference the same subnode, nodes adjacent in the depth-first order are not guaranteed to be on the same disk page. To reduce disk access, frequently referenced nodes, i.e., those with a large in-degree and at lower levels, should be stored in memory, and infrequently referenced nodes stored on disk. However, the exact implementation on disk is transparent to our algorithm. To avoid repeatedly traversing subgraphs caused by multiple edges between two nodes (recall that G is a multi-graph), we assume that G contains at most one edge between two nodes and that one set of labels, denoted $L(w,z)$, is associated with each edge (w,z). The information stored at a node w includes $L(w,z)$ and the list of pointers to z for all subnodes z of w. Therefore, for a simple path $v_1,\ldots,v_n$ in G, the cross product $L(v_1,v_2)\times\ldots\times L(v_{n-1},v_n)$ gives all possible sequences of labels traversed by following the path.

Each *k-generalization*, i.e., a generalization having k leaf nodes, is represented by a reference-tree constructed by k reference-paths of the form $[\&t, l_1, v_1^{j_1},\ldots,l_n,v_n^{j_n}]$, where $\&t$ is a transaction. If we replace $\&t$ in all such paths with the *generic transaction*, denoted $\top$, we still can construct all generalizations constructed before, but we only need to keep one reference-path $[\top, l_1, v_1^{j_1},\ldots,l_n,v_n^{j_n}]$ for all $[\&t, l_1, v_1^{j_1},\ldots,l_n,v_n^{j_n}]$ that differ only in the starting transaction $\&t$. From now on, we assume that this replacement is made. This is analogous to [4] where there is no provision to distinguish supporting transactions for each frequent 1-itemset.

Like Apriori [4], we "grow" a frequent k-generalization using two frequent $(k-1)$-generalizations contained in the k-generalization. Let F_k denote the set of frequent k-generalizations. We use $p_1\ldots p_k$ for both the k-generalization and the reference-tree constructed by reference-paths $p_1,\ldots,p_k$. A superset of F_k, called *k-candidates*, is computed as follows:

Theorem 4.1 Let p_i denote reference-paths. Every frequent k-generalization $p_1\ldots p_k$ in F_k is constructed by two frequent $(k-1)$-generalizations $p_1\ldots p_{k-2}p_{k-1}$ and $p_1\ldots p_{k-2}p_k$ in F_{k-1}.

Since a generalization does not contain the node information necessary for constructing larger generalizations, the construction in Theorem 4.1 should be from reference-trees to reference-trees. In fact, we will keep *all* reference-trees representing a generalization in F_k because we do not know in advance which one will "grow bigger".

The outline of the algorithm: the algorithm has three phases. In Phase I, F_1 is computed by one pass over transactions. In Phase II, in each pass over transactions F_k is computed from F_{k-1}. Heuristics play an essential role to prune the search space. In Phase III, all non-maximally frequent generalizations are removed.

4.1 Phase I: Computing F_1

Consider a reference-path $[\top, l_1, v_1^{j_1},\ldots,l_n,v_n^{j_n}]$. All the information of the 1-generalization represented by the reference-path is coded by the tuple $(l_1,\tau_1,\ldots,l_{n-1},\tau_{n-1},l_n,v_n)$, called the *schema* of the path, where τ_i's denote the types of nodes v_i's, which are 0 for bag and 1 for list. (τ_n is implied by v_n, so not included.) For example, $[\top, l, \&c, l, \&a]$ and $[\top, l, \&e, l, \&a]$ both represent the same 1-generalization $\{l :< l : \&a >\}$ (recall that $\&c$ and $\&e$ have list type). Therefore, the support of 1-generalizations should be associated with the schema of a reference-path, written $sup(l_1, \tau_1, \ldots, l_{n-1}, \tau_{n-1}, l_n, v_n)$, defined as the number of transactions in G from which there is a reference-path with schema $(l_1, \tau_1, \ldots, l_{n-1}, \tau_{n-1}, l_n, v_n)$.

We compute the support of 1-generalizations as follows. For each transaction $\&t$, we depth-first traverse simple paths originating at $\&t$. On visiting a node v_n, if $v_1, \ldots, v_n$ are the non-transaction nodes on the current path with types $\tau_1, \ldots, \tau_n$, we increase all $sup(l_1, \tau_1, \ldots, l_{n-1}, \tau_{n-1}, l_n, v_n)$ by 1 if not increased for $\&t$, where $(l_1, \ldots, l_n)$ is in the cross product $L(\&t, v_1) \times L(v_1, v_2) \times \ldots \times L(v_{n-1}, v_n)$. (The cross product accounts all labels on each edge (v_{i-1}, v_i).) If v_n does not repeat on the current path, we visit all subnodes of v_n in the depth-first manner, regardless of whether visited before (from other supernodes). If v_n repeats on the current path, the traversal of all descendants of v_n is pruned.

Table 2. Computing support of 1-generalizations

Support	$\&t1$	$\&t1, \&t2$	$\&t1, \&t2, \&t3$	$\&t1, \&t2, \&t3, \&t4$	$\&t1, \&t2, \&t3, \&t4, \&t5$
$sup(l, \&c)$	1	1	1	1	1
$sup(l, \&e)$	0	1	2	2	2
$sup(l, 1, l, \&a)$	1	2	3	3	3
$sup(l, 1, l, 1, l, Floor)$	1	2	3	3	3
$sup(l, 1, l, 1, l, Wall)$	1	2	3	3	3
$sup(l, 1, l, Paint)$	1	1	1	1	1
$sup(l, 1, l, Furniture)$	1	2	3	3	3
$sup(l, Photo)$	1	1	2	2	3
$sup(l, Pizza)$	0	1	1	2	2

Example 4.1 Consider Example 1.1. The last column of Table 2 gives the support of 1-generalizations. Refer to Figure 2. For example, the traversal for $\&t1$ visits $\&t1, \&c, \&a, Floor, Wall, Paint, Furniture$ in order and computes the support as in the second column. Subsequently, the traversal for $\&t2$ changes the support to those in the third column. The last column shows the final values of support after all transactions are considered. Even though we have considered the transactions in the order shown, the last column does not depend on such an order.

F_1 consists of all reference-paths whose schemas have the minimum support. To find F_1, we depth-first traverse simple paths originating at each transaction $\&t$. Suppose that $v_1, \ldots, v_n$ are the non-transaction nodes on the current path, with types $\tau_1, \ldots, \tau_n$. For each $(l_1, \ldots, l_n) \in L(\&t, v_1) \times L(v_1, v_2) \ldots \times L(v_{n-1}, v_n)$ such that $sup(l_1, \tau_1, \ldots, l_{n-1}, \tau_{n-1}, l_n, v_n) \geq MINISUP$, if v_n is identical to v_{n-i}, all reference-paths $[\top, l_1, v_1^{j_1}, \ldots, l_n, Ans_i^{j_n}]$ are added to

F_1; otherwise, all reference-paths $[\top, l_1, v_1^{j_1}, \ldots, l_n, v_n^{j_n}]$ are added to F_1. Note that F_1 contains all reference-trees for each frequent 1-generalization.

4.2 Phase II: Computing F_k

The search space. We propose the $(k-1)$-*search trie*, denoted Π_{k-1}, to store F_{k-1}. Π_{k-1} is a tree of maximal depth $k-1$ such that each non-root node represents two things: (a) the frequent reference-path stored at the node and (b) the frequent reference-tree $p_1 \ldots p_j$ where $p_1, \ldots, p_j$ are the reference-paths stored on the path from the root of Π_{k-1} to the node. For $1 \leq j \leq k-1$, F_j is then the set of reference-trees $p_1 \ldots p_j$ in (b). Each pair $p_1 \ldots p_{k-2}p_{k-1}$ and $p_1 \ldots p_{k-2}p_k$ are represented by two sibling leaf nodes at level $k-1$ in Π_{k-1} (with the root at level 0). To start with, Π_1 has one leaf node for each reference-path in F_1.

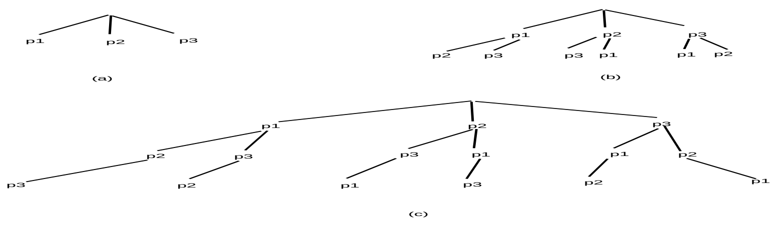

Figure 5. Π_i — level i and above

Generate candidates. To generate k-candidates from Π_{k-1}, for every pair $p_1 \ldots p_{k-2}p_{k-1}$ and $p_1 \ldots p_{k-2}p_k$ represented by sibling leaf nodes l and l', respectively, if none of p_{k-1} and p_k is a prefix of the other, we create a new child for l to represent the k-candidate $p_1 \ldots p_{k-2}p_{k-1}p_k$; the new child is a copy of p_k. This operation is called *extending* l by l'. For example, Figure 5 shows Π_1, Π_2, Π_3 for three reference-paths p_1, p_2, p_3, assuming that all generalizations are frequent.

Compute support. We compute the support of k-candidates in one pass over all transactions. For each transaction $\&t$, we read the defining hierarchy of $\&t$ into memory and depth-first traverse all paths in Π_k starting at the root. Suppose that $p_1, \ldots, p_j$ are the reference-paths on the current path of Π_k. If $j = k$ and generalization $p_1 \ldots p_k$ is weaker than $\&t$, we increase the support at the leaf node representing $p_1 \ldots p_k$. If $j < k$ and $p_1 \ldots p_j$ is not weaker than $\&t$, the traversal into the subtree below can be pruned. This pruning is similar to the hash-tree in [4] used for computing support of itemsets, except that we hash a reference-path, not an item, at at time, our hash is collision-free. After all transactions are read, we delete all leaf nodes at level k in Π_k having support less than the minimum support.

4.3 Pruning the search space

The above candidate generation suffers from constructing too many reference-trees. Two factors contribute to this. Consider the subnodes of a node in a reference-tree. First, different numberings of repeating subnodes, such as $l : v^1, l : u^1, l : v^3$ and $l : v^1, l : u^1, l : v^2$, are considered independently, even though they make no difference in the generalization constructed. The second numbering is *natural* because it uses consecutive numbers $1, 2, 3, \ldots$ for repeating subnodes. Second, different orderings of unordered subnodes, such as $l : v^1, l : u^1, l : v^3$ and $l : u^1, l : v^1, l : v^3$, are considered independently, even though such orderings are not important. The second ordering is *lexicographic*, defined as the order of tuples (l, v, i) for subnodes of the form $l : v^i$.

The idea of pruning the search space is to focus on the natural numbering of repeating subnodes and the lexicographic ordering of subnodes, as defined by canonical reference-trees below. A node in a reference-tree is *lexicographic* if the ordering of its subnodes (if any) is lexicographic unless the node is a squared node. A node in a reference-tree is *natural* if the numbering of its repeating subnodes (if any) is natural. A reference-tree is *lexicographic* (*natural*, resp) if every node is lexicographic (natural, resp). A reference-tree is *canonical* if it is both lexicographic and natural. Testing these properties is straightforward. Since every reference-tree represents the same generalization as some canonical reference-tree, it suffices to construct only canonical reference-trees. However, life is not as straightforward as it sounds: Theo-

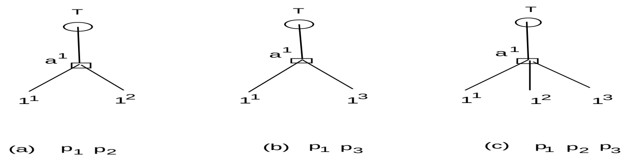

Figure 6. Construct canonical $p_1p_2p_3$ using non-canonical p_1p_3

rem 4.1 requires some non-canonical reference-trees to be constructed. This is shown in Figure 6, where constructing canonical $p_1p_2p_3$, i.e., by extending p_1p_2 by p_1p_3, requires constructing non-canonical p_1p_3 first (whose numbering is non-natural). On the other hand, extending a non-canonical $p_1 \ldots p_{k-1}$

always generates a non-canonical $p_1 \ldots p_k$ because of the non-canonical prefix $p_1 \ldots p_{k-1}$. For the same reason, the result $p_1 \ldots p_k$ cannot generate larger canonical reference-trees.

Pruning 1 $p_1 \ldots p_{k-2}p_{k-1}$ is extended by $p_1 \ldots p_{k-2}p_k$ only if $p_1 \ldots p_{k-2}p_{k-1}$ is canonical and the result $p_1 \ldots p_{k-1}p_k$ is lexicographic.

Pruning 2 After considering all extensions in Π_{k-1}, leaf nodes at level $k-1$ for non-canonical reference-trees can be pruned.

Pruning 3 If $p_1 \ldots p_{k-1}$ is extended into a frequent k-candidate, $p_1 \ldots p_{k-1}$ can be pruned.

In fact, $p_1 \ldots p_{k-1}$ is weaker than the frequent k-candidate generated (thus, non-maximally frequent) and is not useful once becoming a non-leaf node in Π_k. Following Prunings 2 and 3, we revise F_j to be the set of reference-trees represented by unpruned leaf nodes at level j in Π_k, where $j \leq k$. Therefore, Phase III below only needs to examine reference-trees represented by unpruned leaf nodes in Π_k.

Table 3. F_i

reference-trees	supporting transactions	generalizations represented	canonical
		F_1	
$p_1 : [\top, l, Photo]$	$\&t1, \&t3, \&t5$	$\{Photo\}$	
$p_2 : [\top, l, \&c, l, Furniture]$	$\&t1, \&t2, \&t3$	$\{< Furniture >\}$	
$p_3 : [\top, l, \&c, l, \&a]$	$\&t1, \&t2, \&t3$	$\{< \&a >\}$	
$p_4 : [\top, l, \&c, l, \&a, l, Floor]$	$\&t1, \&t2, \&t3$	$\{< \{Floor\} >\}$	
$p_5 : [\top, l, \&c, l, \&a, l, Wall]$	$\&t1, \&t2, \&t3$	$\{< \{Wall\} >\}$	
$p_6 : [\top, l, \&e, l, Furniture^1]$	$\&t1, \&t2, \&t3$	$\{< Furniture >\}$	
$p_7 : [\top, l, \&e, l, Furniture^2]$	$\&t1, \&t2, \&t3$	$\{< Furniture >\}$	no
$p_8 : [\top, l, \&e, l, \&a]$	$\&t1, \&t2, \&t3$	$\{< \&a >\}$	
$p_9 : [\top, l, \&e, l, \&a, l, Floor]$	$\&t1, \&t2, \&t3$	$\{< \{Floor\} >\}$	
$p_{10} : [\top, l, \&e, l, \&a, l, Wall]$	$\&t1, \&t2, \&t3$	$\{< \{Wall\} >\}$	
		F_2	
p_3p_2	$\&t1, \&t2, \&t3$	$\{< \&a, Furniture >\}$	
p_4p_2	$\&t1, \&t2, \&t3$	$\{< \{Floor\}, Furniture >\}$	
p_4p_5	$\&t1, \&t2, \&t3$	$\{< \{Floor, Wall\} >\}$	
p_5p_2	$\&t1, \&t2, \&t3$	$\{< \{Wall\}, Furniture >\}$	
p_8p_6	$\&t1, \&t2, \&t3$	$\{< \&a, Furniture >\}$	
p_8p_7	$\&t1, \&t2, \&t3$	$\{< \&a, Furniture >\}$	no
p_9p_6	$\&t1, \&t2, \&t3$	$\{< \{Floor\}, Furniture >\}$	
p_9p_7	$\&t1, \&t2, \&t3$	$\{< \{Floor\}, Furniture >\}$	no
p_9p_{10}	$\&t1, \&t2, \&t3$	$\{< \{Floor, Wall\} >\}$	
$p_{10}p_6$	$\&t1, \&t2, \&t3$	$\{< \{Wall\}, Furniture >\}$	
$p_{10}p_7$	$\&t1, \&t2, \&t3$	$\{< \{Wall\}, Furniture >\}$	no
		F_3	
$p_4p_5p_2$	$\&t1, \&t2, \&t3$	$\{< \{Floor, Wall\}, Furniture >\}$	
$p_9p_{10}p_6$	$\&t1, \&t2, \&t3$	$\{< \{Floor, Wall\}, Furniture >\}$	
$p_9p_{10}p_7$	$\&t1, \&t2, \&t3$	$\{< \{Floor, Wall\}, Furniture >\}$	no

4.4 Phase III: The maximal phase

Phase III prunes all non-maximally frequent candidates represented by unpruned leaf nodes in Π_k. Observe that a i-generalization can be weaker than a

j-generalization, where i can be less than, equal to, or greater than j. Therefore, it does not work to test only "weaker than" a larger generalization, as for sequential patterns in [5]. Instead, we need to compare the generalization g at each unpruned leaf node in Π_k with generalizations g' in the partial result Max. If g is weaker than g', we discard g immediately; otherwise, we remove all g' in Max that are weaker than g, and add g to Max. Consider the candidates F_1, F_2, F_3 in Table 3. Only two candidates survive the maximal phase, namely, $\{Photo\}$ and $\{< \&a, Furniture >\}$. All other candidates are weaker than some of these two patterns. This verifies that $\{Photo\}$ and $\{< \&a, Furniture >\}$ are the only patterns as claimed in Example 2.1 for MINISUP=3. Table 3 shows the results.

5 Conclusions

The main contribution of this paper is the introduction of a general framework for mining association patterns from semistructured data that are rapidly gaining popularity. In the semistructured world, while many proposals exist on modeling, searching, information exchanging, and structure extracting, data mining is largely unexplored. An important implication of the semistructured nature is that the mining task has to deal with both data and schemas. This requires the data generalization to trace the role of objects and handle arbitrary nesting of ordered and unordered types, and cyclic object references, which is not addressed by the traditional generalization method of dropping fields/items or replacing specialized concepts with general ones. We addressed this issue by proposing a generalization method for an *is-part-of* hierarchy. Based on the new generalization method, we defined the problem of mining association patterns from semistructured data and presented a mining algorithm. We highlighted a wide range of applications of the presented framework. As more and more data do not impose a rigid schema, as those on WWW or digital libraries, we believe that data mining tools dealing with semistructured information are of emerging importance.

References

1. S. Abiteboul, "Querying semi-structured data", ICDT 1997 (http://www-db.stanford.edu/ pub/papers/icdt97.semistructured.ps)
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, J.L. Wiener, "The lorel query language for semistructured data", to appear in Journal of Digital Libraries, 1997 (http://www-db.stanford.edu/pub/papers/lorel96.ps)

3. R. Agrawal, T. Imielinski, A. Swami, "Mining association rules between sets of items in large databases", SIGMOD 1993, 207-216
4. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", VLDB 1994, 487-499
5. R. Agrawal and R. Srikant, "Mining sequential patterns", ICDE 1995, 3-14
6. P. Buneman, S. Davidson, G. Hillebrand, "A query language and optimization techniques for unstructured data", SIGMOD 1996, 505-516
7. W.E. Duellman, L. Trueb, "Biology of amphibians", The Johns Hopkins University Press, 1994
8. S.B. Huffman, C. Baudin, "Toward structured retrieval in semi-structured information spaces", IJCAI 1997, 751-756
9. S. Fortin, L. Liu, "An object-oriented approach to multi-level association rule", CIKM 1996, 65-72
10. T. Fukuda, Y. Morimoto, S. Morishita, "Data mining using two-dimensional optimized association rules: scheme, algorithms, and visualization", SIGMOD 1996, 13-23
11. J. Han and Y. Fu, "Discovery of multiple-level association rules from large databases", VLDB 1995, 420-431
12. M. Holsheimer, M. Kersten, H. Mannila, H. Toivonen, "A perspective on databases and data mining", KDD 1995, 150-155
13. R. Meo, G. Psalia, S. Ceri, "A new SQL-like operator for mining association rules", VLDB 1996, 122-133
14. S. Nestorov, J. Ullman, J. Wiener, S. Chawathe, "Representative objects: concise representations of semistructured, hierarchical data", ICDE 1997 (http://www-db.stanford.edu/pub/papers/representative-object.ps)
15. Y. Papakonstantinuo, H.Garcia-Molina, and J. Widom, "Object exchange across heterogeneous information sources, ICDE 1995, 251-260
16. R. Srikant and R. Agrawal, "Mining generalized association rules", VLDB 1995, 407-419
17. R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables", SIGMOD 1996, 1-12
18. L. Singh, P. Scheuermann, B. Chen, "Generating association rules from semi-structured documents using an extended concept hierarchy", CIKM 1997
19. M. Tresch, N. Palmer, A. Luniewski, "Type classification of semi-structured documents", VLDB 1995, 263-274
20. The Workshop on Management of Semistructured Data 1997, Arizona, http:// www.research.att.com/s̃uciu/workshop-papers.html