1-2005

Ontology-assisted mining of RDF documents

Tao JIANG

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

9

Ontology-Assisted Mining of RDF Documents

Tao Jiang and Ah-Hwee Tan

Summary. Resource description framework (RDF) is becoming a popular encoding language for describing and interchanging metadata of web resources. In this paper, we propose an *Apriori*-based algorithm for mining association rules (AR) from RDF documents. We treat relations (RDF statements) as items in traditional AR mining to mine associations among relations. The algorithm further makes use of a domain ontology to provide generalization of relations. To obtain compact rule sets, we present a generalized pruning method for removing uninteresting rules. We illustrate a potential usage of AR mining on RDF documents for detecting patterns of terrorist activities. Experiments conducted based on a synthetic set of terrorist events have shown that the proposed methods were able to derive a reasonably small set of association rules capturing the key underlying associations.

9.1 Introduction

Resource description framework (RDF) [19, 20] is a data modeling language proposed by the World Wide Web Consortium (W3C) for describing and interchanging metadata about web resources. The basic element of RDF is *statements*, each consisting of a subject, an attribute (or predicate), and an object. A sample RDF statement based on the XML syntax is depicted in Figure 9.1. At the semantic level, an RDF statement could be interpreted as "the subject has an attribute whose value is given by the object" or "the subject has a relation with the object". For example, the statement in Figure 9.1 represents the relation: "Samudra participates in a car bombing event". For simplicity, we use a triplet of the form <subject, predicate, object> to express an RDF statement. The components in the triplets are typically described using an ontology [15], which provides the set of commonly approved vocabularies for concepts of a specific domain. In general, the ontology also defines the taxonomic relations between concepts in the form of a concept hierarchy.

Due to the continual popularity of the semantic web, in a foreseeable future there will be a sizeable amount of RDF-based content available on the web.

```
<rdf:Description about="http://localhost:8080/TerroristOntoEx.rdfs#Samudra ">
<TerroristOntoEx: participate
    rdf:resource = "http://localhost:8080/TerroristOntoEx.rdfs#CarBombing"/>
</rdf:Description>
```

Fig. 9.1. A sample RDF statement based on the XML syntax. "Samudra" denotes the subject, "participate" denotes the attribute (predicate), and "CarBombing" denotes the object.

A new challenge thus arises as to how we can efficiently manage and tap the information represented in RDF documents.

In this paper, we propose a method, known as *Apriori*-based RDF Association Rule Mining (ARARM), for discovering association rules from RDF documents. The method is based on the *Apriori* algorithm [2], whose simplistic underlying principles enable it to be adapted for a new data model. Our work is motivated by the fact that humans could learn useful patterns from a set of similar events or evidences. As an event is typically decomposed into a set of relations, we treat a relation as an item to discover associations among relations. For example, many terrorist attack events may include the scenario that the terrorists carried out a robbery before the terrorist attacks. Though the robberies may be carried out by different terrorist groups and may have different types of targets, we can still derive useful rules from those events, such as "<Terrorist, participate, TerroristAttack> → <Terrorist, rob, CommercialEntity>".

The flow of the proposed knowledge discovery process is summarized in Figure 9.2. First, the raw information content of a domain is encoded using the vocabularies defined in the domain ontology to produce a set of RDF documents. The RDF documents, each containing a set of relations, are used as the input of the association rule mining process. For RDF association rule mining, RDF documents and RDF statements correspond to transactions and items in the traditional AR mining context respectively. Using the ontology, the ARARM algorithm is used to discover generalized associations between relations in RDF documents. To derive compact rule sets, we further present a generalized pruning method for removing uninteresting rules.

The rest of this chapter is organized as follows. Section 9.2 provides a review of the related work. Section 9.3 discusses the key issues of mining association rules from RDF documents. Section 9.4 formulates the problem statement for RDF association rule mining. Section 9.5 presents the proposed ARARM algorithm. An illustration of how the ARARM algorithm works is provided in Section 9.6. Section 9.7 discusses the rule redundancy issue and presents a new algorithm for pruning uninteresting rules. Section 9.8 reports our experimental results by evaluating the proposed algorithms on an RDF

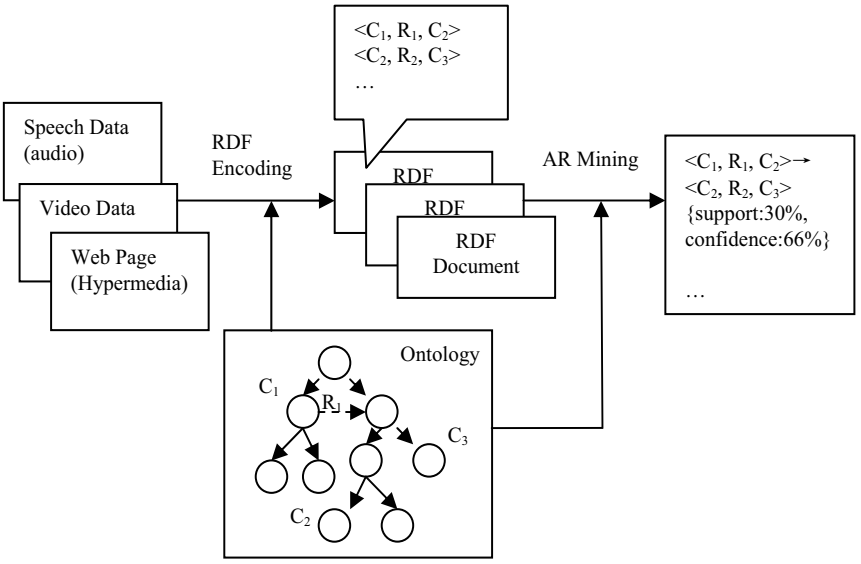

Fig. 9.2. The flow of the proposed RDF association rule mining process.

document set in the Terrorist domain. Section 9.9 concludes and highlights the future work.

9.2 Related Work

Association rule (AR) mining [1] is one of the most important tasks in the field of data mining. It was originally designed for well-structured data in transaction and relational databases. The formalism of typical AR mining was presented by Agrawal and Srikant [2]. Many efficient algorithms, such as *Apriori* [2], Close [16], and FP-growth [10], have been developed. A general survey of AR mining algorithms was given in [12]. Among those algorithms, *Apriori* is the most popular one because of its simplicity.

In addition to typical association mining, variants of the *Apriori* algorithm for mining generalized association rules have been proposed by Srikant and Agrawal [17] to find associations between items located in any level of a taxonomy (*is-a* concept hierarchy). For example, a supermarket may want to find not only specific associations, such as "users who buy the Brand A milk usually tend to buy the Brand B bread", but also generalized associations, such as "users who buy milk tend to buy bread". For generalized rule mining, several optimization strategies have been proposed to speed up support counting. An innovative rule pruning method based on taxonomic information was also provided. Han and Fu [9] addressed a similar problem and presented

an approach to generate frequent itemsets in a top-down manner using an *Apriori*-based algorithm.

In recent years, AR mining has also been used in the field of text mining. Some basic differences between text mining and data mining were described in [8]. Whereas data mining handles relational or tabular data with relatively low dimensions, text mining generally deals with unstructured text documents with high feature dimensions. A framework of text mining techniques was presented in [18]. According to [13], text mining involves two kinds of tasks, namely deductive text mining (DTM) and inductive text mining (ITM). Deductive text mining (or information extraction) involves the extraction of useful information using predefined patterns from a set of text. Inductive text mining, on the other hand, detects interesting patterns or rules from text data. In [5], an AR mining algorithm, known as the Close algorithm, was proposed to extract explicit formal concepts and implicit association rules between concepts with the use of a taxonomy. However, the method was designed to discover *statistical* relations between concepts. It therefore can not be used to extract *semantic* relations among concepts from unstructured text data.

Recently, some interesting work on mining semi-structured XML data has been reported [3, 4, 6, 7, 14]. A general discussion of the potential issues in applying data mining to XML was presented in [4]. XML is a data markup language that provides users with a syntax specification to freely define elements, to describe their data and to facilitate data exchange on the web. However, the flexibility has resulted in a heterogeneous problem for knowledge discovery on XML. Specifically, XML documents that describe similar data content may have very different structures and element definitions. In [14], this problem was discussed and a method for determining the similarity between XML documents was proposed. In contrast to relational and transaction databases, XML data have a tree structure. Therefore, the context for knowledge discovery in XML documents should be redefined. Two approaches for mining association rules from XML documents have been introduced [3, 7]. In general, both approaches aimed to find similar nested element structures among the branches of the XML Document Object Model (DOM) trees [21]. At the semantic level, the detected association rules represent the correlation among attributes (nested elements) of a certain kind of elements. In [6], an approach was presented that used association rule mining methods for detecting patterns among RDF queries. The detected association rules were then used to improve the performance of RDF storage and query engines. However, the method was designed for mining association rules among subjects and attributes, but not among RDF statements.

9.3 Mining Association Rules from RDF

RDF/RDFS data consist of a set of RDF statements in the form of triplets. The RDF triplets form a directed graph (RDF Graph) with labels (attributes or predicates) on its edges. For the purpose of data exchange, RDF/RDFS uses an XML-based syntax. Mining association rules from RDF/RDFS data presents a number of unique challenges, described as follows.

First, each RDF statement is composed of a subject, an attribute (or predicate), and an object, that are described using the vocabularies from a predefined domain ontology. Suppose the ontology includes 100 concepts and an average of three predicates between each pair of concepts, the number of possible RDF statements is already 30,000. In real applications, the number of concepts defined in domain ontology could far exceed 100. Therefore, the number of distinct statements may be so large that each single RDF statement only appears a very small number of times, far below the typical minimum support threshold. This motivates our approach in mining generalized association rules.

Second, RDF statements with the same attributes can be generalized, if both their subjects and objects share common super-concepts. Recursively generalizing a set of statements creates a relation lattice. The information in the relation lattices can be used to improve the performance of itemset candidate generation and frequency counting (see Section 9.5).

Third, in contrast to items in relational databases, statements in RDF documents may be semantically related. Intuitively, semantically related statements should be statistically correlated as well. This motivates us to define a new interestingness measure for pruning uninteresting rules.

Furthermore, RDF statements express a rich set of explicit semantic relations between concepts. This makes the association rules discovered from RDF documents more understandable for humans.

9.4 Problem Statement

The problem formulation of association rule mining on RDF documents is given as follows. As we are interested in mining the associations among RDF statements, i.e., relations, we will use the term "relationset" instead of "itemset" in our description.

Let $\boldsymbol{O} = <\boldsymbol{E}, \boldsymbol{S}, \boldsymbol{H}>$ be an ontology, in which $\boldsymbol{E}=\{e_1,e_2,\ldots,e_m\}$ is a set of literals called entities; $\boldsymbol{S}=\{s_1,s_2,\ldots,s_n\}$ is a set of literals called predicates (or attributes); and $\boldsymbol{H}$ is a tree whose nodes are entities. An edge in H represents an ***is-a*** relationship between two entities. If there is an edge from e_1 to e_2, we say e_1 is a parent of e_2, denoted by $e_1 > e_2$; and e_2 is a child of e_1, denoted by $e_2 < e_1$. We call e+ an ***ancestor*** of e, if there is a path from e+ to e in $\boldsymbol{H}$, denoted by e+ >> e. If e >> e_1, e >> e_2....e>> e_k, we call e a ***common ancestor*** of $e_1,e_2,\ldots,e_k$, denoted by e >> $e_1,e_2,\ldots,e_k$. For a set of entities

$e_1, e_2, \ldots e_k$, if $e' \in \{ e \mid e >> e_1, e_2, \ldots e_k \}$ and not exists $e'' \in \{ e \mid e >> e_1, e_2, \ldots e_k \}$ such that $e' >> e''$, e' is called the ***least common ancestor*** of $e_1, e_2, \ldots e_k$, denoted by $e' = \text{lca}(e_1, e_2, \ldots e_k)$.

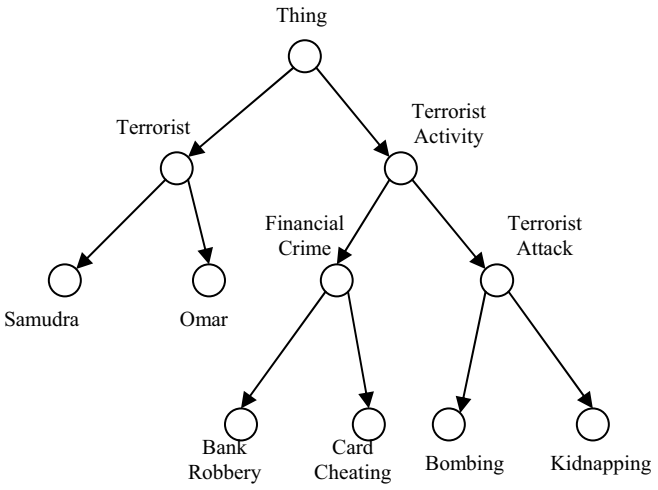

Fig. 9.3. A simple concept hierarchy for the Terrorist domain ontology.

Table 9.1. A sample RDF knowledge base ***SD*** in the terrorist domain.

Transaction	Relation
1	<Samudra, raiseFundBy, BankRobbery > < Samudra, participate, Bombing>
2	<Omar, raiseFundBy, CardCheating> <Omar, participate, Kidnapping>
3	<Omar, participate, Bombing>

Typically, there is a top-most entity in the ontology, called ***thing***, which is the ancestor of all other entities in $\boldsymbol{E}$. Thus, $\boldsymbol{E}$ and $\boldsymbol{H}$ define a concept hierarchy. A sample concept hierarchy for the Terrorist domain is shown in Figure 9.3. The ontology $\boldsymbol{O}$ defines a set of vocabularies for describing knowledge in a specific domain.

Let $\boldsymbol{D}$ be a set of *transactions*, called a ***knowledge base***. Each ***transaction*** T is a set of *relations* (RDF statements), where each ***relation*** r is a triplet in the form of <x, s, y>, in which x, y$\in \boldsymbol{E}$, and s$\in \boldsymbol{S}$. We call x the ***subject*** of the relation r, denoted by sub(r)=x; we call s the ***predicate*** of

the relation r, denoted by pred(r)=s; we call y the ***object*** of the relation r, denoted by obj(r) = y.

A sample knowledge base ***SD*** in the terrorist domain is shown in Table 9.1. There are three transactions in the knowledge base, each of which contains a set of relations describing a terrorist event.

A set of relations $R = \{r_1, r_2, \ldots, r_d\}$ (where $r_i =< x_i, s_i, y_i >$ for $i = 1, \ldots, d$) is called an ***abstract relation*** of $r_1, r_2, \ldots, r_d$ in $\boldsymbol{D}$, if and only if $s_1 = s_2 = \ldots = s_d$ and there exist e' and $e'' \in \boldsymbol{E}$, such that e'=lca$\{x_1, x_2, \ldots, x_d\}$, $e'' = lca\{y_1, y_2, \ldots, y_d\}$, $e' \neq$ ***thing***, and $e'' \neq$ ***thing***, and not exist $r' =< x', s', y' >$ in transactions of $\boldsymbol{D}$ where $r' \notin R, s' = s_1 = ... = s_d, e' >> x'$ and $e'' >> y'$. We also define the subject of R as sub$(R) = e' = lca\{x_1, x_2, \ldots, x_d\}$; the predicate of R as pred(R) $= s'$, where $s' = s_1 = s_2 \ldots = s_d$; and the object of R as obj$(R) = e'' = lca\{y_1, y_2, \ldots, y_d\}$. For simplicity, we use the triplet $< e', s', e'' >$, similar to that for denoting relations, to represent abstract relations. We call an abstract relation R a ***sub-relation*** of an abstract relation R', if $R \subset R'$ hold. An abstract relation R is the ***most abstract relation***, if and only if there does not exist another abstract relation R' in $\boldsymbol{D}$ where R$\subset R'$.

We say a transaction T supports a relation r if $r \in T$. We say a transaction T supports an abstract relation R if R$\cap T \neq \emptyset$. We assume that each transaction T has an id, denoted by *tid*. We use $r.tids = \{tid_1, tid_2, \ldots, tid_n\}$ to denote the set of ids of the transactions in $\boldsymbol{D}$ that support the relation r. We define the ***support*** of r, denoted by support$(r) = |r.tids|$. Similarly, for an abstract relation R, we define R.$tids = \cup r.tids$, for all $r \in$ R. We further define the ***support*** of R, denoted by support$(R) = |R.tids|$. In this paper, we use A, B, or C to represent a set of abstract relations $\{R_1, R_2 \ldots R_n\}$, named ***relationset***. We define the ***support*** of a relationset A, denoted by support$(A) = | \cap R_i.tids|, i = 1, 2, \ldots n$. We call a relationset A a ***frequent relationset***, if support(A) is greater than a user-defined minimum support (minSup). An ***association rule*** in $\boldsymbol{D}$ is of the form A$\rightarrow$B, where A, B, and A$\cup$B are frequent relationsets and its ***confidence***, denoted by confidence(A$\rightarrow$B) = support(A$\cup$B)/support(A), is greater than a user-defined minimum confidence (minConf).

9.5 The ARARM Algorithm

Following the method presented in [2], our *Apriori*-based approach for mining association rules can be decomposed into the following steps.

1. Find all 1-frequent relationsets. Each 1-frequent relationset contains only one abstract relation R, which may contain one or more relations $\mathrm{r}_1 \ldots \mathrm{r}_n$ ($n \geq 1$).
2. Repeatedly generate k-frequent ($k \geq 2$) relationsets based on $k-1$-frequent relationsets, until no new frequent relationsets could be generated.
3. Generate association rules and prune uninteresting rules.

9.5.1 Generation of 1-Frequent Relationsets

For generating 1-frequent relationsets, we use a top-down strategy. We first find all the most abstract relations by scanning the RDF knowledge base ***D*** and merging similar relations that have common abstract relations. Next, we repeatedly split the frequent abstract relations into their sub-relations until all abstract relations are not frequent. We then keep all the frequent abstract relations as the 1-frequent relationsets. The procedure of identifying most abstract relations is summarized in Figure 9.4.

Algorithm 1: Find Most Abstract Relations
Input: A set of transactions ***D***
Output: A set of most abstract relations
(1) Rlist:=Φ
(2) **for each** transaction $T \in \boldsymbol{D}$ do
(3) **for each** relation r in T do
(4) **if** (exist an abstract relation R in Rlist **AND**
(5) (r ∈ R **OR** (pred(r) == pred(R) **AND** lca(sub(R), sub(r)) != ***thing*** **AND**
(6) lca(obj(R), obj(r)) != ***thing***)))
(7) R := {r} ∪ R //For the set of transactions ***D'*** scanned, R is an abstract relation.
(8) R.r.tids := R.r.tids ∪ {T.tid} //In R, R.r.tids represents the set of ids of the transactions that have been scanned and support r. If r doesn't exist in R before, R.r.tids represents Φ.
(9) **else**
(10) R' :={r} //if r could not be merged into an existing abstract relation, create a new abstract relation R' for r.
(11) R'.r.tids := {T.tid}
(12) Rlist :={R'} ∪ Rlist
(13) Output Rlist

Fig. 9.4. The algorithm for identifying most abstract relations.

Through the algorithm defined in Figure 9.4, we obtain a set of most abstract relations (*Rlist*). Each abstract relation and its sub-relations form a relation lattice. An example of a relation lattice is shown in Figure 9.5. In this lattice, <Terrorist, participate, TerroristAttack> is the most abstract relation subsuming the eight relations at the bottom levels. The middle-level nodes in the lattice represent sub-abstract-relations. For example, <Samudra, participate, Bombing> represents a sub-abstract-relation composed of two relations, namely <Samudra, participate, CarBombing> and <Samudra, participate, SuicideBombing>.

The algorithm for finding all 1-frequent relationsets is given in Figure 9.6. For each most abstract relation R in Rlist, if R is frequent, we add R into 1-frequent relationsets L_1 and we traverse the relation lattice whose top vertex is R to find all 1-frequent sub-relations of R (Figure 9.6a).

Figures 9.6b and 9.6c define the procedures of searching the abstract relation lattice. First, we recursively search the right children of the top relation to find 1-frequent relationsets and add them into L_1. Then, we look at each

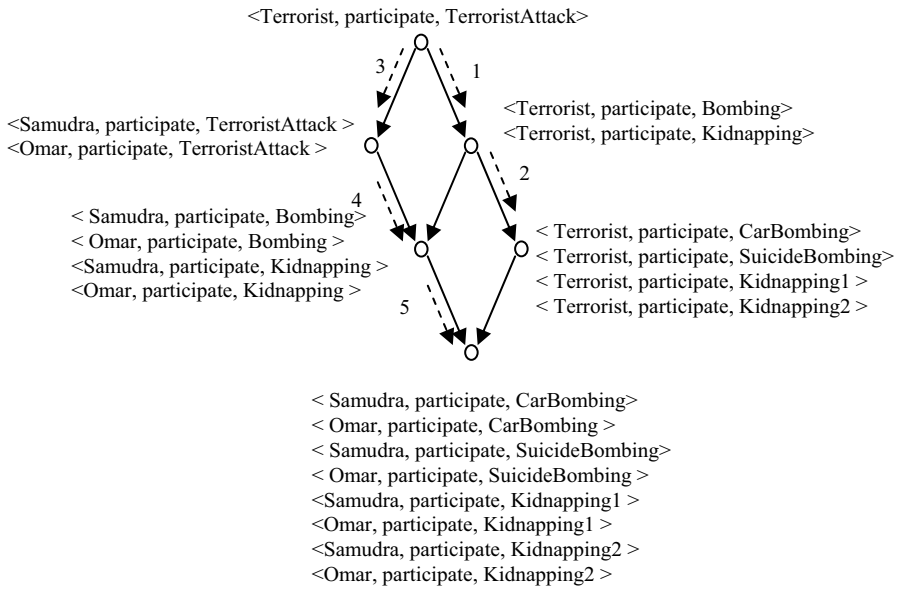

Fig. 9.5. The flow of searching in a sample relation lattice.

left child of the top abstract relation. If it is frequent, we add it into L_1 and recursively search the sub-lattice using this left child as the new top relation. In Figure 9.5, the dashed arrows and the order numbers of the arrows illustrate the process of searching the lattice for 1-frequent relationsets.

Here, we define the notions of right/left children, right/left sibling, and left/right parent of an abstract relation in a relation lattice. In Figure 9.5, <Terrorist, participate, Bombing> and <Terrorist, participate, Kidnapping> are sub-relations of <Terrorist, participate, TerroristAttack>. They are derived from their parent by drilling down its object based on the domain concept hierarchy. We call them the ***right children*** of <Terrorist, participate, TerroristAttack> and call <Terrorist, participate, TerroristAttack> the ***left parent*** of <Terrorist,participate, Bombing> and <Terrorist, participate, Kidnapping>. Similarly, if some sub-relations are derived from their parent by drilling down its subject, we call them the ***left children*** of their parent and call their parent the ***right parent*** of these sub-relations. If there exists an abstract relation that has a left child A and a right child B, A is called a ***left sibling*** of B and B is called a ***right sibling*** of A.

Lemma 1. ***(Abstract Relation Lattice)*** *Given an abstract relation* $R = <x, s, y>$ *with a right parent* $R_{rp} = <x+, s, y>$ *(or left parent* $R_{lp} = <x, s, y+>$*), if* $support(R_{rp}) < minSup$ *(or* $support(R_{lp}) < minSup$*), it can be derived that* $support(R) < minSup$.

Algorithm 2: Find 1-frequent relationsets
Input: A set of transactions ***D***
Output: A set of 1-frequent relationsets
(1) marList := getMostAbsRelations(***D***)
(2) **for each** most abstract relation R in marList do
(3) **if** (support(R) $\geq$ minSup)
(4) L_1 := {R} $\cup$ L_1
(5) L_1':=searchAbsRelationLattice(R, NULL); //NULL means that most abstract relations don't have right siblings.
(6) L_1 := L_1 $\cup$ L_1'
(7) Output L_1

(a)

Procedure searchAbsRelationLattice
Input: Abstract relation **R**; hash table that stores right siblings of R, **rSiblings**
Output: 1-frequent relationsets in the relation lattice of R (excluding R)
(1) L_1':= Φ
(2) L_1' :=searchRightChildren(R, rSiblings)
(3) **for each** left children R_{lc} of R do //get left child by drilling down the subject of R
(4) **if** support(R_{lc}) $\geq$ minSup
(5) L_1' := { R_{lc} } $\cup$ L_1'
(6) R_{lc}.rightParent := R
(7) R.leftChildren.insert(R_{lc})
(8) L_1'':=searchAbsRelationLattice(R_{lc}, R.rightChildren)
(9) L_1' := L_1' $\cup$ L_1''
(10) Output L_1'

(b)

Procedure searchRightChildren
Input: Abstract relation **R**; hash table that stores right siblings of R, **rSiblings**
Output: 1-frequent relationsets among the right descendants of R
(1) L_1^R:= Φ
(2) **for each** right children R_{rc} of R do
(3) rParent := getRParent(R_{rc}, rSiblings) //get the right parent of R_{rc} by finding the right sibling of R that has the same object with R_{rc}.
(4) **if** support(rParent) < minSup
(5) **continue;** //Optimization 1.
(6) **if** support(R_{rc}) $\geq$ minSup
(7) **if** rParent != NULL
(8) rParent.leftChildren.insert(R_{rc});
(9) R_{rc}.rightParent := rParent
(10) R.rightChildren.insert(R_{rc})
(11) R_{rc}.leftParent := R
(12) L_1^R := { R_{rc} } $\cup$ L_1^R
(13) L_1^R':=searchRightChildren(R_{rc}, rParent.rightChildren)
(14) L_1^R := L_1^R $\cup$ L_1^R'
(15) Output L_1^R

(c)

Fig. 9.6. The algorithm for generating 1-frequent relationsets.

Proof. We only need to prove support(R) $\leq$ support(R_{rp}) (and support(R) $\leq$ support(R_{lp})). Since R is a sub-relation of R_{rp} (or R_{lp}), for each relation r$\in$R, r$\in R_{rp}$(or r$\in R_{lp}$) holds. Therefore, $\cup$r.tids (r$\in$R) is a subset of $\cup r'$.tids ($r' \in R_{rp}$ or $r' \in R_{lp}$). Then the cardinality of $\cup$r.tids is smaller than or equals to the cardinality of $\cup r'$.tids, i.e. support(R) $\leq$ support(R_{rp}) (support(R) $\leq$ support(R_{lp})).

According to Lemma 1, once we find that the left parent or right parent of an abstract relation is not frequent, we do not need to calculate the support of this abstract relation and can simply prune it away. This forms our *Optimization Strategy 1.*

9.5.2 Generation of *k*-Frequent Relationsets

Observation 1. Given two abstract relations R_1 and R_2, if $R_1 \cap R_2 \neq \emptyset$ and $|R_1| \geq |R_2|$, either R_2 is a sub-abstract-relation of R_1 (i.e. $R_1 \cap R_2 = R_2$), or R_1 and R_2 have a common sub-abstract-relation R_3 in the relation lattice (i.e. $R_1 \cap R_2 = R_3$). For example, in Figure 9.5, two abstract relations <Samudra, participate, TerroristAttack> and <Terrorist, participate, Kidnapping> have a common sub-abstract-relation <Samudra, participate, Kidnapping> = {<Samudra, participate, Kidnapping1>, <Samudra, participate, Kidnapping2>}.

Lemma 2. *Given a* k*-relationset* $A=\{R_1, R_2, \ldots, R_k\}$*, if there are two abstract relations* R_i *and* R_j *(*$1 \leq i, j \leq k$ *and* $i \neq j$*), such that* $|R_i| \geq |R_j|$ *and* $R_i \cap R_j \neq \emptyset$*, there exists a* k$-1$*-relationset B with support(B) = support(A).*

Proof. According to Observation 1, there exists an abstract relation R', where either $R'=R_j$ or R' is a common sub-abstract-relation of R_i and R_j ($R_i \cap R_j = R'$). Therefore, there exists a $k-1$-relationset B = A$\cup\{R'\}$ <minus> $\{R_i, R_j\}$ and support(B) = support(A).

According to Lemma 2, a k-relationset that includes two intersecting abstract relations is redundant and should be discarded. This is the basis of our *Optimization Strategy 2.*

Observation 2. Given two 2-frequent relationsets A={R_1, R_2} and B={R_1, R_2+}, where R_1, R_2, and R_2+ are frequent abstract relations and R_2+ is an ancestor of R_2, if the support of the relationset {R_1, R_2} equals the support of the relationset {R_1, R_2+}, the relationset B is redundant because A and B are supported by the same set of transactions. As {R_1, R_2} provides a more precise semantics than {R_1, R_2+}, the latter is redundant and should be discarded. This is *Optimization Strategy 3.*

The procedure of generating k-frequent relationsets L_k is described in Figure 9.7. To generate L_k, we need to first generate k-candidate relationsets based on $k-1$-frequent relationsets. We search the $k-1$-frequent relationset

Algorithm 3: Find k-frequent relationsets
Input: 1-frequent relationset list $\mathbf{L_1}$
Output: k-frequent relationsets($k \geq 2$)
(1) k:=2
(2) L:= Φ
(3) **while** $|L_{k-1}| \geq k$ do
(4) C_k:= generateCandidate(L_{k-1})
(5) **for each** candidate relationset $A \in C_k$ do
(6) **if** support(A) ≥ minSup
(7) L_k={A} U L_k
(8) prune(L_k) //Optimization 3
(9) L := L ⋃ L_k
(10) k := k+1
(11) Output L

Fig. 9.7. The algorithm for identifying k-frequent relationsets.

pair (A, B), where A, B ∈ L_{k-1}, A={R_1,R_2,...,R_{k-1}}, B={R'_1,R'_2,...,R'_{k-1}}, R_i= R'_i (i=1,2,..., $k-2$), and $R_{k-1} \cap R'_{k-1} = \emptyset$ (*Optimization Strategy 2*). For each such pair of k-1-frequent relationsets (A, B), we generate a k-candidate relationset A∪B={R_1,R_2,...,R_{k-1},R'_{k-1}}. We use C_k to denote the entire set of k-candidate relationsets. We further generate L_k by pruning the k-candidate relationsets whose supports are below minSup. In L_k, some redundant k-frequent relationsets also need to be removed according to *Optimization Strategy 3*.

9.5.3 Generation of Association Rules

For each frequent relationset A, the algorithm finds each possible sub-relationset B and calculates the confidence of the association rule B → A < minus > B, where A < minus > B denotes the set of relations in A but not in B. If confidence(B→A<minus>B) is larger than minConf, B→A<minus>B is generated as a rule.

9.6 Illustration

In this section, we illustrate our ARARM algorithm by mining associations from the sample knowledge base ***SD*** depicted in Table 9.1. Suppose that the minimum support is 2 and the minimum confidence is 66%. The relations (RDF statements) in the knowledge base are constructed using the ontology as shown in Figure 9.3. The predicate set is defined as ***S*** = {raiseFundBy, participate}.

First, we aggregate all relations in ***SD*** (as described in Figure 9.4) and obtain two most-abstract relations (Table 9.2). Because the supports of those two abstract relations are all greater than or equal to minimum support of 2, they will be used in the next step to generate 1-frequent relationsets.

Table 9.2. The most-abstract relations obtained from the knowledge base ***SD***.

Most-Abstract Relations	Support
<Terrorist, raiseFundBy, FinancialCrime>	2
<Terrorist, participate, TerroristAttack>	3

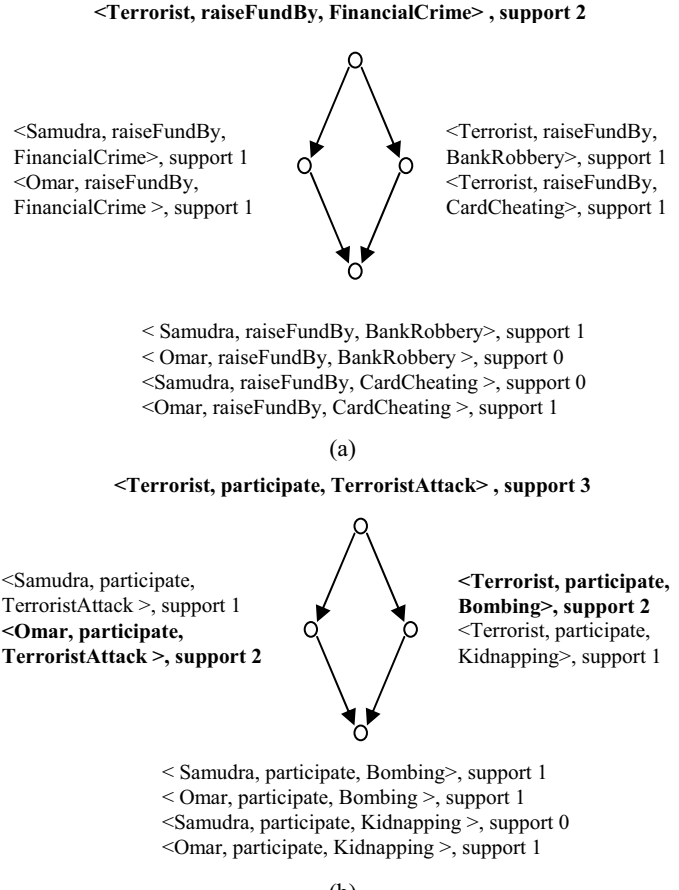

Fig. 9.8. The relation lattices of the two most-abstract relations.

Next, we search the relation lattices to find 1-frequent relationsets. The relation lattices of the two most-abstract relations are shown in Figure 9.8.

In Figure 9.8a, because all of the relations in the second level are below the minimum support, the relations at the bottom of the lattice will not be considered. In Figure 9.8b, because the relations <Omar, participate, TerroristAttack> and <Terrorist, participate, Bombing> are frequent, their child relation <Omar, participate, Bombing> at the bottom of the lattice will still be considered. Other relations will be directly pruned because either the support of their left parent or right parent is below the minimum support.

Table 9.3. The 1-frequent relationsets identified from the sample knowledge base ***SD***.

1-Frequent Relationsets	Support
{<Terrorist, raiseFundBy, FinancialCrime>}	2 {1,2}
{<Terrorist, participate, TerroristAttack>}	3 {1,2,3}
{<Omar, participate, TerroristAttack >}	2 {2,3}
{<Terrorist, participate, Bombing>}	2 {1,3}

Table 9.4. The k-frequent relationsets ($k \geq 2$) identified from the sample knowledge base ***SD***.

k-Frequent Relationsets	Support
{<Terrorist, raiseFundBy, FinancialCrime>, <Terrorist, participate, TerroristAttack> }	2{1,2}

Table 9.5. The association rules discovered from the sample knowledge base ***SD***.

Association rules	Support/ Confidence
<Terrorist, raiseFundBy, FinancialCrime> →<Terrorist, participate, TerroristAttack>	2/66.6%
<Terrorist, participate, TerroristAttack> →<Terrorist, raiseFundBy, FinancialCrime>	2/100%

After traversing the relation lattices, we obtain the 1-frequent relationsets as shown in Table 9.3. Using the k-frequent relationset generation algorithm,

we obtain the k-frequent relationsets ($k \geq 2$), depicted in Table 9.4. The association rule generation algorithm then derives the two association rules as shown in Table 9.5.

9.7 Pruning Uninteresting Rules

Association rule mining algorithms typically produce a large number of rules. Therefore, efficient methods for detecting and pruning uninteresting rules are usually needed. A general survey on rule interestingness measures was presented in [11]. In [13], a set of commonly used properties for defining the interestingness of the associations were introduced. The issues of pruning redundant rules with the use of a concept hierarchy were discussed in [9] and [17]. Srikant and Agrawal presented a method for calculating the expected support and confidence of a rule according to its "ancestors" in a concept hierarchy. A rule is considered as "redundant" if its support and confidence can be estimated from those of its "ancestors". The method however assumes that the items appearing in an association are independent.

For mining association rules among the relations in RDF documents, the problem of measuring interestingness becomes more complex on two accounts. First, generalization and specialization of RDF relations are more complicated. For example, a relation may have two direct parents in the relation lattice. Second, the relations may be semantically related. For example, the relations <Samudra, raiseFundBy, BankRobbery> and <Samudra, participate, Bombing> refer to the same subject *Samudra*. They are thus more likely to appear together than two unrelated relations. To improve upon Srikant's method [17], we develop a generalized solution for calculating the expected support and confidence of a rule based on its ancestors.

We call a relationset A+ an ***ancestor*** of relationset A if A+ and A have the same number of relations and A+ can be derived from A by replacing one or more concepts in A with their ancestors in a concept hierarchy. Given an association rule A→B, we call the association rules A+→B, A+→B+, and A→B+, the ***ancestors*** of A→B. We call A+→B+ a close ancestor of A→B, if there does not exist a rule $A' \rightarrow B'$ such that $A' \rightarrow B'$ is an ancestor of A→B and A+→B+ is an ancestor of $A' \rightarrow B'$. A similar definition applies to both A+→B and A→B+.

For calculating the expected support and confidence of an association rule based on its close ancestors' support and confidence, the contribution of the concept replacement could be estimated according to the three cases described below.

- Concept replacement in both the left- and right-hand sides. For example, an association rule AR1: <a, rel1, b> → <c, rel2, a> could be derived from an association rule AR2: <a+, rel1, b> → <c, rel2, a+> by replacing concept "a+" with its sub-concept "a". This kind of concept replacement

only influences the support of the association rule. The expected support and confidence of AR1 is given by

$$support_E(AR1) = support(AR2) \cdot P(a|a+) \tag{9.1}$$

and

$$confidence_E(AR1) = confidence(AR2) \tag{9.2}$$

where P(a|a+) is the conditional probability of a, given a+.

- Concept replacement in the left-hand side only. For example, an association rule AR1: <a, rel1, b> → <c, rel2, d> could be generated from an association rule AR2: <a+, rel1, b> → <c, rel2, d> by replacing the concept "a+" with its sub concept "a". This kind of concept replacement influences only the support of the association rule. We can calculate the support and confidence of AR1 by using Eqns. (9.1) and (9.2).
- Concept replacement in the right-hand side only. For example, an association rule AR1: <c, rel1, d> → <a, rel2, b> could be generated from an association rule AR2: <c, rel1, d> → <a+, rel2, b> by replacing concept "a+" with its sub concept "a". This kind of concept replacement influences both the support and the confidence of the association rule. We can calculate the expected support and confidence of AR1 by

$$support_E(AR1) = support(AR2) \cdot P(a|a+) \tag{9.3}$$

and

$$confidence_E(AR1) = confidence(AR2) \cdot P(a|a+) \tag{9.4}$$

respectively.

Note that the above three cases may be combined to calculate the overall expected support and confidence of an association rule. The conditional probability P(a|a+) can be estimated by the ratio of the number of the leaf sub-concepts of "a" and the number of the leaf sub-concepts of "a+" in the domain concept hierarchy. For example, in Figure 9.3, the number of the leaf sub-concepts of "Financial Crime" is two and the number of the leaf sub-concepts of "Terrorist Activity" is four. The conditional probability P(Financial Crime |Terrorist Activity) is thus estimated as 0.5.

Following the idea of Srikant and Agrawal [17], we define the interestingness of a rule as follows. Given a set of rules S and a minimum interest factor F, a rule A→B is ***interesting***, if there is no ancestor of A→B in S or both the support and confidence of A→B are at least F times the expected support and confidence of its close ancestors respectively. We name the above interestingness measure *expectation measure with semantic relationships* (EMSR). EMSR may be used in conjunction with other pruning methods, such as those described in [13].

9.8 Experiments

Experiments were conducted to evaluate the performance of the proposed association rule mining and pruning algorithms both quantitatively and qualitatively. Our experiments were performed on an IBM T40 (1.5GHz Pentium Mobile CPU, 512MB RAM) running Windows XP. The RDF storage system was Sesame (release 1.0RC1) running on MySQL database (release 4.0.17). The ARARM algorithm was implemented using Java (JDK 1.4.2).

(1) Every event includes an RDF relation <Terrorist, participate, TerroristActivity>.

(2) 90% of the events, which include an RDF relation <Terrorist, participate, Bombing> also include an RDF relation <Terrorist, participate, Robbery>.

(3) 85% of the events include an RDF relation <Terrorist, takeVehicle, Vehicle>.

(4) For any event containing an RDF relation <Terrorist, participate, SuicideBombing >, if it also includes (probability of 85%) <Terrorist, takeVehicle, Vehicle>, there is a probability of 80% that <Terrorist, takeVehicle, Vehicle> is in specialized form <Terrorist, takeVehicle, Truck>.

(5) 85% of the events include an RDF relation <Terrorist, useWeapon, Weapon>.

(6) For any event containing <Terrorist, participate, Bombing>, if it also includes (probability of 85%) <Terrorist, useWeapon, Weapon>, there is a probability of 100% that <Terrorist, useWeapon, Weapon> is in specialized form <Terrorist, useWeapon, Bomb>, and there is a probability of 70% that <Terrorist, useWeapon, Weapon> is in specialized form <Terrorist, useWeapon, PlasticBomb>.

(7) For any event containing an RDF relation <Terrorist, participate, Kidnapping >, if it also includes (probability of 85%) <Terrorist, useWeapon, Weapon>, there is a probability of 100% that <Terrorist, useWeapon, Weapon> is in specialized form <Terrorist, useWeapon, NormalWeapon>, and there is a probability of 90% that <Terrorist, useWeapon, Weapon> is in specialized form <Terrorist, useWeapon, AK-47>.

Fig. 9.9. The seven domain axioms for generating the terrorist events.

Due to a lack of large RDF document sets, we created a synthetic data set, which contained a large number of RDF statements related to the terrorist domain. The data set has enabled us to conduct empirically extensive experiments of the various algorithms. The ontology for encoding terrorist events contained a total of 44 concepts (including classes and instances) and four predicates (attributes). Among the four predicates, three were used for describing the relationships between concepts in the terrorist events and one was used to provide additional information, such as the start time of terrorist events. To perform empirical evaluation, 1000 RDF documents were generated using a set of domain axioms (Figure 9.9). The maximum number of RDF statements in a single RDF document was four. We then performed as-

sociation rule mining according to the ARARM algorithm and evaluated if the extracted rules captured the underlying associations specified by the domain axioms. With a 5% minimum support and a 50% minimum confidence, the ARARM algorithm generated 76 1-frequent and 524 k-frequent ($k \geq 2$) relationsets, based on which 1061 association rules were extracted. With a 10% minimum support and a 60% minimum confidence, the algorithm produced 42 1-frequent relationsets, 261 k-frequent relationsets, and 516 association rules.

We observed that although the events were generated based on only seven domain axioms, a much larger number of rules were extracted. For example, axiom 2 may cause the association rule "<Terrorist, participate, Bombing> → <Terrorist, participate, Robbery>" to be generated. Axiom 2 may also result in the association rule "<Terrorist, participate, Robbery> → <Terrorist, participate, Bombing>", as <Terrorist, participate, Bombing> tended to co-occur with <Terrorist, participate, Robbery>. In addition, axioms can be combined to generate new rules. For example, axioms 1, 3, and 5 can combine to generate association rules, such as "<Terrorist, participate, TerroristActivity> → <Terrorist, takeVehicle, Vehicle>, <Terrorist, useWeapon, Weapon>". As the association rule sets generated using the ARARM algorithm may still be quite large, pruning methods were further applied to derive more compact rule sets.

We experimented with a revised version of Srikant's interestingness measure method [17] and the EMSR method for pruning the rules. The experimental results are summarized in Table 9.6 and Table 9.7. We further experimented with two simple statistical interestingness measure methods [13] described below:

- *Statistical correlations measure (SC):* Given a rule R1→R2, where R1 and R2 are relationsets, if the conjunctive probability $P(R1,R2) \neq P(R1) \cdot P(R2)$, R1 and R2 are correlated and the rule R1→R2 is considered as interesting.
- *Conditional independency measure (CI):* Given two rules R1→R2 and R1, R3→R2 where R1, R2 and R3 are relationsets, if the conditional probability $P(R2|R1) = P(R2|R1,R3)$, we say R2 and R3 are conditionally independent and the rule R1, R3→R2 is considered as redundant and uninteresting.

Table 9.6. The experimental results using Srikant's method.

minSup/ minConf	Number of rules before pruning	Number of rules after applying Srikant's method	Number of rules after combining with SC and CI
5%/50%	1061	297	148
10%/60%	516	162	72

Table 9.7. The experimental results using the EMSR interestingness measure method.

minSup/ minConf	Number of rules before pruning	Number of rules after applying EMSR	Number of rules after combining with SC and CI
5%/50%	1061	277	91
10%/60%	516	177	46

When pruning association rules, we first applied Srikant's and the EMSR methods on the rule sets produced by the ARARM algorithm and derived association rule sets considered as interesting for each strategy. Then we combined Srikant's method and the EMSR method individually with the SC and CI interestingness measures to derive even smaller rule sets.

We observed that there was no significant difference between the numbers of rules obtained using the EMSR method and Srikant's method. However, by combining with other pruning methods, the resultant rule sets of EMSR were about 40% smaller than those produced by Srikant's method. The reason was that the rule sets produced by Srikant's method contained more rules similar to those produced using the SC and CI measures. In other words, Srikant's method failed to remove those uninteresting rules that could not be detected by the SC and CI measures.

For evaluating the quality of the rule sets produced by the EMSR method, we analyzed the association rule set obtained using a 5% minimum support and a 50% minimum confidence. We found that the heuristics of all seven axioms were represented in the rules discovered. In addition, most of the association rules can be traced to one or more of the domain axioms. A representative set of the association rules is shown in Table 9.8.

9.9 Conclusions

We have presented an *Apriori*-based algorithm for discovering association rules from RDF documents. We have also described how uninteresting rules can be detected and pruned in the RDF AR mining context.

Our experiments so far have made use of a synthetic data set, created based on a set of predefined domain axioms. The data set has allowed us to evaluate the performance of our algorithms in a quantitative manner. We are in the process of building a real Terrorist data set by annotating web pages.

Our ARARM algorithm assumes that all the RDF relations of interest could fit into the main memory. In fact, the maximum memory usage of our algorithm is proportional to the number of relations. When the number of

Table 9.8. Sample association rules obtained by ARARM and EMSR.

Examples of association rules discovered	Explanation	Domain axioms
<Terrorist, participate, Kidnapping>→ <Terrorist, useWeapon, AK-47> {support:0.166; confidence:0.817}	The rule reflects the heuristics of a domain axiom directly.	7
<Terrorist, useWeapon, AK-47>→ <Terrorist, participate, Kidnapping> {support:0.166; confidence:0.790}	The rule reflects the heuristics of a domain axiom indirectly.	7
<Terrorist, participate, Kidnapping>→ <Terrorist, useWeapon, Gun> {support:0.168; confidence:0.827}	The rule is a generalized form of a domain axiom.	7
<Terrorist, useWeapon, PlasticBomb>→ <Terrorist, participate, Robbery> {support:0.251; confidence:0.916}	The rule reflects the interaction of two or more domain axioms.	2, 6
<terroristA, participate,TerroristActivity >→ <terroristA, useWeapon, Weapon>{support:0.051; confidence:0.809}	The rule is generated due to spurious events. The support for this type of rule is usually very low.	

relations is extremely large, an optimization strategy should be developed to maintain the efficiency of the AR mining process.

For simplicity, we assume that the subjects and objects of the RDF statements in the document sets are in the form of RDF Unified Resource Identifier (URI), each referring to a term defined in a domain ontology. According to the RDF/RDFS specification [19, 20], an RDF statement could also include RDF literals and blank nodes. We will address these issues in our future work.

References

[1] Agrawal, R., T. Imielinski and A. Swami, 1993: Mining association rules between sets of items in large databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 207–16.

[2] Agrawal, R., and R. Srikant, 1994: Fast algorithms for mining association rules. *Proceedings of the 20th International Conference in Very Large Databases*, 487–99.

[3] Braga, D., A. Campi, S. Ceri, M. Klemettinen and P.L. Lanzi, 2003: Discovering interesting information in XML data with association rules. *Proceedings of ACM Symposium on Applied Computing*, 450–4.

[4] Buchner, A. G., M. Baumgarten, M. D. Mulvenna, R. Bohm and S. S. Anand, 2000: Data mining and XML: Current and future issues. *Proceedings of International Conference on Web Information Systems Engineering 2000 IEEE,* **II**, 131–5.

[5] Cherif Latiri, Ch. and S. Ben Yahia, 2001: Generating implicit association rules from textual data. *Proceedings of ACS/IEEE International Conference on Computer Systems and Applications*, 137–43.

[6] Ding, L., K. Wilkinson, C. Sayer and H. Kuno, 2003: Application-specific schema design for storing large RDF datasets. *First International Workshop on Practical and Scalable Semantic Systems.*

[7] Ding, Q., K. Ricords and J. Lumpkin, 2003: Deriving general association rules from XML data. *Proceedings of International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 348–52.

[8] Dorre, J., P. Gerstl and R. Seiffert, 1999: Text mining: Finding nuggets in mountains of textual data. *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 398-401.

[9] Han, J., and Y. Fu, 1995: Discovery of multi-level association rules from large databases. *Proceedings of the 21st International Conference in Very Large Databases*, 420–31.

[10] Han, J., J. Pei and Y. Yin, 2000: Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM-SIGMOD International Conference on Management of Data*, 1–12.

[11] Hilderman, R., J., and H. J. Hamilton, 1999: Knowledge discovery and interestingness measures: A survey. *Technical Report CS 99-04*, Department of Computer Science, University of Regina.

[12] Hipp, J., U. Guntzer and G. Nakaeizadeh, 2000: Algorithms for association rule mining: A general survey and comparison. *ACM SIGKDD Explorations*, **2(1)**, 58–64.

[13] Kodratoff, Y., 2001: Rating the interest of rules induced from data and within texts. *Proceedings of Database and Expert Systems Applications 12th International Conference*, 265–9.

[14] Lee, J.-W., K. Lee and W. Kim, 2001: Preparations for semantics-based XML mining. *Proceedings of 1st IEEE International Conference on Data Mining*, 345–52.

[15] Maedche, A., and V. Zacharias, 2002: Clustering ontology-based metadata in the semantic web. *Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 342–60.

[16] Pasquier, N., Y. Bastide, R. Taouil and L. Lakhal, 1998: Pruning closed itemset lattices for association rules. *Proceedings of the BDA French Conference on Advanced Databases*, 177–96.

[17] Srikant, R., and R. Agrawal, 1995: Mining generalized association rules. *Proceedings of the 21st International Conference in Very Large Databases*, 407–19.

[18] Tan, A.-H., 1999: Text mining: The state of the art and the challenges. *Proceedings of the Pacific Asia Conference on Knowledge Discovery and Data Mining PAKDD'99 workshop on Knowledge Discovery from Advanced Databases*, 65–70.
[19] W3C, RDF Specification. URL: `www.w3.org/RDF/`.
[20] W3C, RDF Schema Specification. URL: `www.w3.org/TR/rdf-schema/`.
[21] XML DOM Tutorial. URL: `www.w3schools.com/dom/default.asp`.