9-2006

Mining RDF metadata for generalized association rules

Tao JIANG

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

Mining RDF Metadata for Generalized Association Rules

Tao Jiang and Ah-Hwee Tan

School of Computer Engineering
Nanyang Technological University, Nanyang Avenue, Singapore 639798
{jian0006, asahtan}@ntu.edu.sg

Abstract. In this paper, we present a novel frequent generalized pattern mining algorithm, called *GP-Close*, for mining generalized associations from RDF metadata. To solve the *over-generalization* problem encountered by existing methods, GP-Close employs the notion of *generalization closure* for systematic *over-generalization reduction*. Empirical experiments conducted on real world *RDF* data sets show that our method can substantially reduce pattern redundancy and perform much better than the original generalized association rule mining algorithm *Cumulate* in term of time efficiency.

1 Introduction

Resource Description Framework (RDF)[1] is a specification proposed by the World Wide Web Consortium (W3C) for describing and interchanging semantic metadata on the Semantic Web [1]. 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, offering tremendous opportunities in discovering useful knowledge from large RDF databases.

The basic element of RDF is RDF statements, each consisting of a subject, a predicate, and an object. For simplicity, we use a triplet of the form <subject, predicate, object> to express a RDF statement. Based on RDF, RDF Schema[2] (RDFS) is further proposed for defining vocabulary definitions. A RDF vocabulary defines a set of types (RDFS classes) and predicates (RDF properties) for describing web resources. Taxonomic relations between classes (properties) can also be defined. Given a set of RDF vocabularies that defines a set of taxonomies of RDFS classes, generalized association rule mining can be applied to RDF documents for discovering generalized associations between RDF statements. The discovered associations may have applications including optimizing RDF storage and query processing, enhancing information source recommendation in Semantic Web, and association-based classification or clustering of web resources.

Generalized association rule mining (GARM) [2] extends association rule mining [3] by exploiting item taxonomies in the mining task. Given a set of items $\mathcal{I}$

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/rdf-schema/

S. Bressan, J. Küng, and R. Wagner (Eds.): DEXA 2006, LNCS 4080, pp. 223–233, 2006.

and a large database of transactions $\mathcal{D}$, where each transaction is a set of items $T \subseteq \mathcal{I}$ with a unique identifier *tid*, an association rule is an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ (called itemsets or patterns) and $X \cap Y = \emptyset$. A taxonomy $\mathcal{T}$ is defined as an acyclic directed graph on the set of all items $\mathcal{I}$. An edge in $\mathcal{T}$ from i_1 to i_2 represents an "*is-a*" relationship. GARM aims to discover rules spanning items across different levels of taxonomies.

The unique characteristics of RDF data sets lie in the large sizes of RDF documents and the complex structures of the RDF statement hierarchies wherein a RDF statement can be generalized in many ways, by generalizing its subject, object, predicate, or their combinations (see Figure 1(b) in subsection 3.2). In GARM, *frequent pattern mining* (FPM) is usually the most time-consuming part. For RDF data sets, existing GARM algorithms extracting all possible frequent generalized patterns do not work efficiently due to the fact that a large portion of the frequent generalized patterns is *over-generalized*. Processing over-generalized patterns can seriously increase the computation cost and degrade the performance of the mining algorithms. In this paper, we present a novel algorithm, called *GP-Close* (Closed Generalized Pattern Mining), for mining frequent generalized patterns from RDF metadata with *full over-generalization reduction*. We employ the notion of *generalization closure* to formulate over-generalization reduction as a *closed pattern mining* problem [4] [5].

The rest of the paper is organized as follows. Section 2 introduces the related work. Section 3 formalized the problem of mining frequent generalized patterns from RDF metadata and highlights the over-generalization problem. Section 4 presents the GP-Close algorithm. Section 5 reports our experiments based on two real world RDF data sets. Concluding remarks are given in the final section.

2 Related Work

Generalized association rule mining was first proposed in [2], where a family of algorithms, namely *Basic*, *Cumulate*, *Stratify*, *Estimate* and *EstMerge*, was reported. *Basic* is almost a direct application of Apriori algorithm [6] to the extended transaction databases. *Cumulate* extends Basic by employing three optimization strategies: filtering the useless ancestors in transactions, pre-computing items' ancestors, and pruning itemsets containing an item and its ancestors. Based on Cumulate, further optimizations are involved in *Stratify*, *Estimate*, and *EstMerge*. However, they only perform marginally better than Cumulate.

In [7], *Prutax* algorithm makes use of tidset-intersection for support counting to avoid multiple database scans. It adopts a right-most and depth-first search (DFS) strategy for itemset enumeration. Prutax further guarantees that the generalized itemsets are always generated before their specialized itemsets. The downward closure property and taxonomy information are also used for candidate pruning. More recently, an algorithm, *SET* [8], adopted a *tax-based* and *join-based* strategy for itemset enumeration. Nevertheless, this strategy may cause problems when the taxonomies used in the mining process are not tree-structured. In such cases, duplicated patterns may be generated.

A common limitation of the above algorithms is that they generate all the frequent patterns including over-generalized ones. On RDF data sets, most of the extracted patterns may be over-generalized. Calculation of over-generalized patterns can seriously increase computation cost. Over-generalization problem is first identified in [9] when mining generalized substructure in connected graphs. Though [9] adopted a pruning strategy to remove some over-generalized patterns, it did not provide a solution for full over-generalization reduction.

3 Mining Frequent Generalized Patterns from RDF Data

3.1 Problem Statement

We define the frequent generalized pattern mining task based on a simplified view of RDF model as follows.

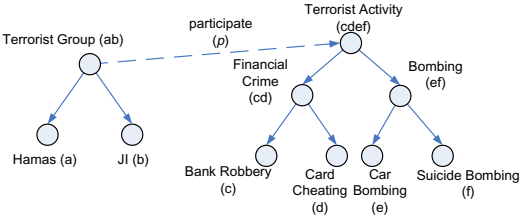
Definition 1. *Let* $\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\}$ *defines an* **RDF vocabulary**, *where* $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ *is a set of entity identifiers;* $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ *is a set of predicate identifiers; and* $\mathcal{H}$ *is a directed acyclic graph. An edge in* $\mathcal{H}$ *represents an* **is-a** *relationship between two entities(See Figure 1(a)). If there is an edge from* e_1 *to* e_2*, we say* e_1 *is a parent of* e_2 *and* e_2 *is a child of* e_1*. We call* $\hat{e}$ *an* **ancestor** *of* e*, if there is a path from* $\hat{e}$ *to* e*. The function,* domain*:* $\mathcal{P} \rightarrow 2^{\mathcal{E}}$*, relates a predicate to a set of entities that can be its subject. The function,* range*:* $\mathcal{P} \rightarrow 2^{\mathcal{E}}$*, relates a predicate to a set of entities that can be its object.*

The above definition simplifies the RDF model in two aspects:

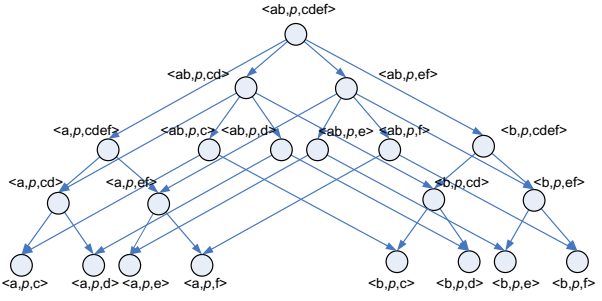
- We treat instances and RDF classes both as entities. Correspondingly, we treat "rdf:type" and "rdfs: subClassOf" predicates as "is-a" relation without discrimination. By this way, we can integrate instances and RDF classes into one taxonomy so that the mining task can be simplified.
- We do not consider the hierarchy of RDF properties (predicate hierarchy) at the current stage. However, the predicate hierarchy can be easily incorporated into the generalized association mining framework.

Definition 2. *Given a RDF vocabulary* $\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\}$*, we define a* **relation** *(RDF statement)* r *on* $\mathcal{V}$ *as a triplet* $<x, p, y>$*, where* $x, y \in \mathcal{E}$*,* $p \in \mathcal{P}$*,* $x \in domain(p)$*, and* $y \in range(p)$*. We call* x*,* p*, and* y *the* **subject***, the* **predicate***, and the* **object** *of* r*. A relation* $\hat{r} = <x_1, p_1, y_1>$ *is called a* **generalized relation** *(***ancestor***) of another relation* $r = <x_2, p_2, y_2>$*, if and only if: (1)* $\hat{r} \neq r$*, (2)* $p_1 = p_2$*, (3)* x_1 *is an ancestor of* x_2 *or* $x_1 = x_2$ *and (4)* y_1 *is an ancestor of* y_2 *or* $y_1 = y_2$*. We use* $\mathcal{R}^{\mathcal{V}}$ *to denote the set of all relations on* $\mathcal{V}$*. Relations in* $\mathcal{R}^{\mathcal{V}}$ *and their generalization/specialization relationships form a* **relation hierarchy**, $H_r^{\mathcal{V}}$ *(see Figure 1(b)).*

Definition 3. *A* **relationset** *(***pattern***) is a set of relations* $X \subseteq \mathcal{R}^{\mathcal{V}}$ *(*X *does not contain both a relation and its ancestor). We call* X *a* **generalized relationset** *of another relationset* Y *and* Y *a* **specialized relationset** *of* X*, if and*



(a) A sample RDF vocabulary.



(b) Generalized relation hierarchy.

Pattern ***X***:
{<Terrorist Group, *participate*, Financial Crime>, <Terrorist Group, *participate*, Car Bombing>}

Generalization Closure of ***X***:
{<Terrorist Group, *participate*, Financial Crime>, <Terrorist Group, *participate*, Car Bombing>, <Terrorist Group, *participate*, Bombing>, <Terrorist Group, *participate*, Terrorist Activity>}

(c) An illustration of generalization closure.

Fig. 1. Elements of RDF data sets

only if: (1)$X \neq Y$*, (2)*$\forall r \in X, \exists r^* \in Y$ *such that* $r = r^*$ *or* r *is an ancestor of* r^* *and (3)* $\forall r^* \in Y, \exists r \in X$ *such that* $r = r^*$ *or* r *is an ancestor of* r^**. Given a set of RDF documents* $\mathcal{D}$*, where each document consists of a set of relations, the* **support** *of a relationset* X *(*$supp(X)$*) is defined as the proportion of RDF documents containing* X *or a specialized relationset of* X*.*

Based on the above definitions, given a set of RDF documents, our task of mining frequent generalized patterns is to extract *frequent* relationsets (patterns) whose supports are larger than a predefined *minimum support* (*minsup*).

3.2 Over-Generalization and Mining Closed Generalization Closures

We explain the over-generalization problem using an example. Figure 1(a) shows a RDF vocabulary $\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\}$, where $\mathcal{E} = \{a, b, c, d, e, f, ab, cd, ef, cdef\}$, $\mathcal{P} = \{p\}$, $dom(p) = \{a, b, ab\}$, and $range(p) = \{c, d, e, f, cd, ef, cdef\}$. Figure 1(b) shows the relation hierarchy containing all relations in $\mathcal{R}^{\mathcal{V}}$. A sample RDF database $\mathcal{D}$ on $\mathcal{V}$ is shown in Table 1. Given $minsup = 50\%$, all frequent

Table 1. A sample RDF database

ID	RDF Documents
1	<a, *p*, d> <a, *p*, e>
2	<b, *p*, c> <b, *p*, e>
3	<a, *p*, f>
4	<b, *p*, f>

Table 2. Frequent generalized relationsets in sample RDF database

Support	Frequent Generalized Relationsets (*minsup*=50%)
50%	{<a, *p*, ef>} {<a, *p*, cdef>} {<ab,*p*,cd>} {<ab, *p*, e>} {<ab, *p*, f>} {<b, *p*, ef>} {<b, *p*, cdef>} {<a, *p*, cdef> <ab, *p*, ef>} {<ab, *p*, cd> <ab, *p*, e>} {<ab, *p*, cd> <ab, *p*, ef>} {<ab, *p*, ef> <b, *p*, cdef>}
100%	{<ab, *p*, ef>} {<ab, *p*, cdef>}

generalized relationsets are listed in Table 2. Note that some patterns look quite similar, e.g. {<ab, *p*, cd>, <ab, *p*, e>} and {<ab, *p*, cd>, <ab, *p*, ef>}. In fact, the second pattern is a generalization of the first one and they have the same support (50%). Intuitively, with the same support, a specialized pattern should be more interesting than its generalizations as the information conveyed by the specialized pattern is more precise. Therefore, the second pattern is redundant. Based on this observation, we define over-generalization as follows:

Definition 4. *A frequent relationset X is* **over-generalized** *if there exists a specialized relationset Y of X with $supp(X) = supp(Y)$.*

In Table 2, six (46%) frequent patterns (underlined) are *over-generalized*. In real-world RDF data sets, the proportion of over-generalized patterns may be much higher than this. For reducing redundant over-generalized patterns, we propose an innovative approach based on the notion of generalization closures.

Definition 5. *Given a RDF vocabulary $\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\}$ and $\mathcal{R}^{\mathcal{V}}$ on $\mathcal{V}$, we define a function φ_{gc} on $2^{\mathcal{R}^{\mathcal{V}}}$: $\varphi_{gc}X = \bigcup_{r \in X} \mathcal{G}(r)$, where $X \subset \mathcal{R}^{\mathcal{V}}$ and $\mathcal{G}(r)$ is a set of relations that contains r and all its generalized relations. φ_{gc} is a* closure operator *[10], called* **generalization closure operator**. *And $\varphi_{gc}X$ is called the* **generalization closure** *of X.*

A sample generalization closure of a pattern X is shown in Figure 1(c). We can prove that φ_{gc} is a *closure operator* and all generalization closures form a *closure system* [10] with the three properties: $X \subseteq Y \Rightarrow \varphi_{gc}X \subseteq \varphi_{gc}Y$ (monotony), $X \subseteq \varphi_{gc}X$ (extensity), and $\varphi_{gc}\varphi_{gc}X = \varphi_{gc}X$ (idempotency).

Five useful lemmas related to generalization closures are listed below. The first three lemmas are intuitive. For the space limitation, we only give the proofs of Lemma 4 and 5.

Lemma 1. *Given a relationset X and a RDF database $\mathcal{D}$, $supp(X) = supp(\varphi_{gc}X)$.*

Lemma 2. *Given X is a generalized relationset of Y, $\varphi_{gc}X \subset \varphi_{gc}Y$ holds.*

Lemma 3. *Given two generalization closures $\varphi_{gc}X$ and $\varphi_{gc}Y$ $(X, Y \subseteq \mathcal{R}^{\mathcal{V}})$, $\varphi_{gc}X \cup \varphi_{gc}Y$ is also a generalization closure $(\varphi_{gc}(X \cup Y))$.*

Lemma 4. *Given a frequent relationset X, if $\varphi_{gc}X$ is closed (i.e. there is not a $\varphi_{gc}Y \supset \varphi_{gc}X$ where $supp(\varphi_{gc}X) = supp(\varphi_{gc}Y)$), X is not over-generalized.*

Proof. Suppose X is over-generalized. It follows that there is a specialized pattern Y of X, where $supp(Y) = supp(X)$. According to Lemma 1 and 2, easy to see that $\varphi_{gc}(X) \subset \varphi_{gc}(Y)$ and $supp(\varphi_{gc}(X)) = supp(\varphi_{gc}(Y))$, i.e. $\varphi_{gc}(X)$ is not closed. This is contradictory to the statement that $\varphi_{gc}(X)$ is closed.

Lemma 5. *The support of all frequent relationsets can be derived from the set of all frequent closed generalization closures.*

Proof. As each relationset X has a generalization closure $\varphi_{gc}(X)$ with the same support, the support of X can be derived from $\varphi_{gc}(X)$ by the following ways:

1. If $\varphi_{gc}(X)$ is a closed closure, $supp(X) = supp(\varphi_{gc}(X))$.
2. Otherwise, there exists a closed closure $\varphi_{gc}(Y) \supset \varphi_{gc}(X)$ and does not exist a closed closure $\varphi_{gc}(Z)$ with $\varphi_{gc}(X) \subset \varphi_{gc}(Z) \subset \varphi_{gc}(Y)$. It follows that $supp(X) = supp(\varphi_{gc}(X)) = supp(\varphi_{gc}(Y))$.

Lemma 4 and 5 motivate us to discover all *closed generalization closures* for over-generalization reduction.

4 GP-Close Algorithm

Based on Lemma 4 and 5, we propose an algorithm, called GP-Close (Closed Generalized Pattern Mining), for discovering all closed generalization closures. Comparing with existing GARM algorithms, such as Cumulate, our algorithm discover closed generalization closures but not frequent generalized patterns. However, Lemma 5 shows that all frequent generalized patterns can be easily derived from the output of the GP-Close algorithm.

4.1 Closure Enumeration and Sorting

Lemma 3 guarantees that we can gradually generate larger closures by merging smaller ones. Based on this, GP-Close adopts a depth-first closure enumeration method using an enumeration tree. The pseudo-code of GP-Close algorithm is presented in Algorithm 1 and 2. GP-Close first calculates all 1-frequent relationsets. Then, the generalization closures of the 1-frequent relationsets are created and sorted (see Algorithm 1). A *support-increasing and length-decreasing* strategy is adopted for closure sorting. This implies that the specialized closures will be enumerated first. Later in this section, we will discuss this in more details.

Figure 2 shows the closure enumeration process based on the RDF database in Table 1. Initially, the enumeration tree contains only the root closure, i.e. the

Algorithm 1. GP-Close

Input:
RDF database: $\mathcal{D}$
Generalized relation lookup table: GRT
Support Threshold: $minsup$
Output:
The set of all closed frequent generalization closures: $\mathcal{C}$
1: $ce\_tree.root = \emptyset$ //initialize closure enumeration tree
2: $ce\_tree.root.supp = 1$
3: $ce\_tree.gc\_list = \{\varphi_{gc}\{r\} | r \in \mathcal{R}^{\mathcal{V}} \wedge supp(r) \geqslant minsup\}$ ////Constructing child closure set of $ce\_tree.root$ from 1-frequent RDF statements by looking up GRT
4: Sort($ce\_tree.gc\_list$) //sort closures (length-decreasing/support-increasing)
5: Closure-Enumeration(ce_tree, $\mathcal{C} = \emptyset$)
6: return $\mathcal{C}$

empty set with the support of 100%, and a set of closures of 1-frequent relationsets as children of the root (see Algorithm 1). Then, for each child of the root closure, we can expand it by merging it with other child closures. For example, in Figure 2, the closure $\{<ab, p, f>, <ab, p, ef>, <ab, p, cdef>\}$ is combined with the closure $\{<ab, p, cd>, <ab, p, cdef>\}$ to generate a larger closure $\{<ab, p, f>, <ab, p, ef>, <ab, p, cd>, <ab, p, cdef>\}$. Using this child closure as the root and the newly generated closures as its children, a sub closure enumeration tree is constructed (Algorithm 2 line 7 - 22). We can see that all descendants of a (sub) enumeration tree are expansions of the tree root. If a (sub) tree root can be subsumed by a discovered closed closure, i.e. it is not closed, traversing this (sub) tree cannot generate new closed generalization closures. Therefore, the (sub) tree can be pruned (Algorithm 2 line 1). For example, in Figure 2, the closure $\{<ab, p, cd>, <ab, p, cdef>\}$ is subsumed by the closed closure $\{<ab, p, f>, <ab, p, ef>, <ab, p, cd>, <ab, p, cdef>\}$. As a result, the corresponding sub closure enumeration tree is pruned. The following are three cases in which one closure can subsume another:

1. A *specialized closure* can subsume a generalized closure.
2. A closure can be subsumed by one of its *super relationsets* (*closures*).
3. A closure can be subsumed by a *super set of its specialized closures*.

Our specialization-first sorting strategy increases the occurrences of subsumption cases (1) and (3), i.e. there is a larger probability that a later constructed enumeration tree can be pruned.

The function *Prune* (Algorithm 2 line 2) performs two tasks. One is removing infrequent closures. Another is checking the subsumption among children of the current enumeration tree root and eliminating the closures that can be subsumed. In Figure 2, $\{<b, p, cdef>, <ab, p, cdef>\}$ can be subsumed by $\{<b, p, ef>, <b, p, cdef>, <ab, p, ef>, <ab, p, cdef>\}$. Thus it can be pruned.

The function *Closed-Closure* (Algorithm 2 line 3) generates the closed generalization closure. It finds all child closures that have the same support as the

Algorithm 2. Closure-Enumeration

Input:
Closure enumeration tree: ce_tree
A set of discovered closed frequent generalization closures: $\mathcal{C}$
Output:
The expanded set of closed frequent generalization closures: $\mathcal{C}$

1: If $\exists c^* \in \mathcal{C}$ where c^* subsumes n, return $\mathcal{C}$. //Subtree Pruning
2: Prune($ce\_tree.gc\_list$)
3: c = Closed-Closure(ce_tree)
4: $\mathcal{C} = \mathcal{C} \cup \{c\}$ //if c is not subsumed by another closed closure $c^* \in \mathcal{C}$
5: $candidates = \emptyset$
6: **for** each closure $gc_i \in ce\_tree.gc\_list$ **do**
7: $ce\_tree^*.root = gc_i$; $ce\_tree^*.gc\_list = \emptyset$ //initialize a sub enumeration tree
8: **for** each $gc_j \in ce\_tree.gc\_list$, with $i < j$ **do**
9: $gc^* = gc_i \cup gc_j$
10: **if** $gc_i.tidset \neq null$ and $gc_j.tidset \neq null$ **then**
11: $tidset = gc_i.tidset \cap gc_j.tidset$
12: **if** tidset-buffer is not overflow **then**
13: $gc^*.tidset = tidset$; $gc^*.supp = |tidset|$
14: **else**
15: $gc^*.supp = |tidset|$
16: **end if**
17: **else**
18: $candidates = candidates \cup \{gc^*\}$
19: **end if**
20: $ce\_tree^*.gc\_list = ce\_tree^*.gc\_list \cup \{gc^*\}$
21: **end for**
22: If $candidates \neq \emptyset$, perform $hash\text{-}counting(candidates)$.
23: Closure-Enumeration(ce_tree^*, $\mathcal{C}$)
24: **end for**
25: return $\mathcal{C}$

root of current closure enumeration tree. We call such child closures ***support-undescending expansions*** of the root closure (or simply *undescending expansions*). Then, these undescending expansions are merged to form a closed closure. In Figure 2, there exist two undescending expansions $\{<ab, p, ef>$, $<ab, p, cdef>\}$ and $\{<ab, p, cdef>\}$ of root closure $\{\}$, so that $\{<ab, p, ef>$, $<ab, p, cdef>\}$ is extracted as a closed closure. If there is no such undescending expansion, the root will be extracted as a closed closure.

4.2 Hybrid Support Counting

Some existing association rule mining algorithms propose to use the *transaction ID set* (*tidset*) for itemset support counting [5]. However, in real life applications, tidsets may not be able to fit into the physical memory. A hybrid counting strategy is thus designed in GP-Close for handling data sets under different circumstances. It allows users to define a tidset buffer with a maximum buffer size. The support counting is initially performed by scanning databases (DB). During DB scans, the algorithm tries to build tidsets for candidate closures if these tidsets can fit into the pre-located buffer. The constructed tidsets are then used for subsequent tidset based support counting (Algorithm 2 line 10-17).

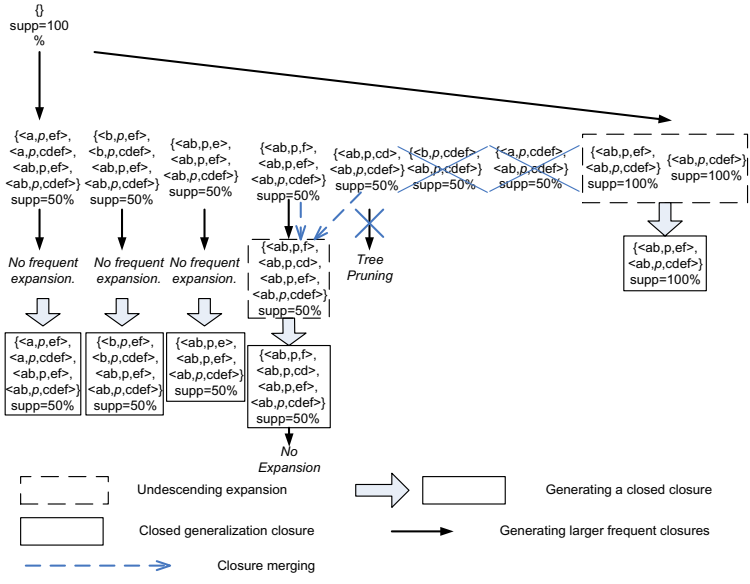


Fig. 2. Closure Enumeration

5 Experiments

The GP-Close algorithm was implemented using Java (JDK 1.4.2). GP-Close with different sizes of tidset buffer were used in our experiments, namely GP-Close-*0*, GP-Close-*500*, and GP-Close-*50000*, with a tidset buffer of 0 KB, 500 KB, and 50,000 KB respectively. We also implemented the Cumulate algorithm [2] as a reference of performance comparison.

5.1 Data Sets

Our experiments are conducted on two real world RDF data sets, namely the foafPub data set provided by the UMBC eBiquity Research Group and the ICT-CB data set extracted from an online database of International Policy Institute for Counter-Terrorism. foafPub is a set of RDF files that describes peoples and their relationships with the use of the FOAF vocabulary[3]. ICT-CB documents are descriptions of car bombing events. The statistics of the RDF vocabularies and RDF documents in the two data sets are summarized in Table 3.

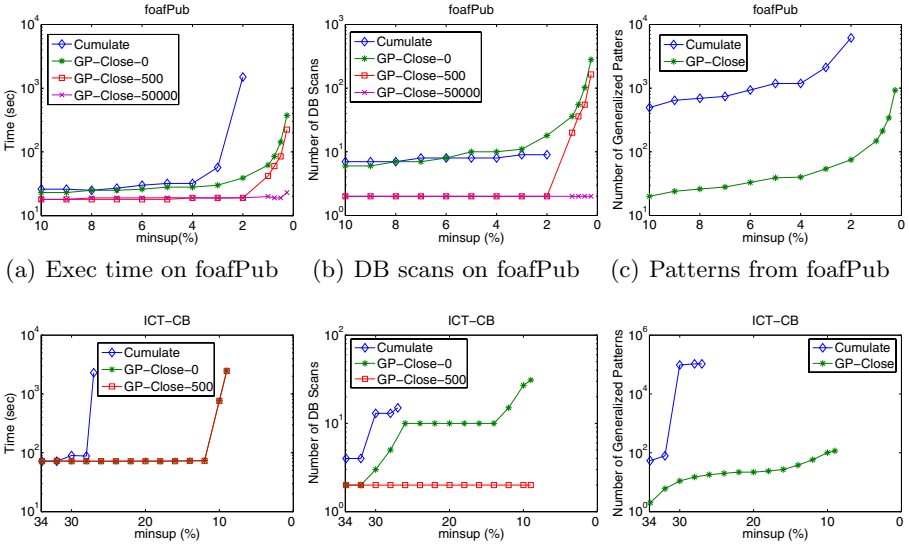
5.2 Performance Study

Figure 3(a) and 3(d) show the computation time of the algorithms with respect to minimum support. We find that Cumulate can work properly only with high *minsup*. When the *minsup* is high, the performance of the algorithms are comparable. The GP-Close-*0* is slightly slower than other versions of GP-Close due

[3] http://xmlns.com/foaf/0.1/

Table 3. Statistics of the foafPub and ICT-CB Data Sets. N_d - number of RDF documents, N_r - number of RDF relations, N_r^* - number of distinct RDF relations, N_{gr} - number of distinct generalized relations.

RDF Data set	RDF property	RDF class	Instance	Avg. Fanout	N_d	N_r	N_r^*	N_{gr}	Min len.	Max len.	Avg. len.
foafPub	36	6801	66613	13	6170	85778	83759	207119	1	3188	14
ICT-CB	53	1806	2546	3	127	2224	1814	175020	1	104	17



(a) Exec time on foafPub (b) DB scans on foafPub (c) Patterns from foafPub

(d) Exec time on ICT-CB (e) DB scans on ICT-CB (f) Patterns from ICT-CB

Fig. 3. Performance of GP-Close compared with Cumulate

to the fact that it involves more IO accesses. When the *minsup* is low, all three versions of GP-Close algorithm perform more than an order of magnitude faster than Cumulate. This is because the CPU computation becomes the bottleneck of the algorithms as the number of frequent patterns increases.

Figure 3(b) and 3(e) show the number of DB scans performed by the algorithms. The overall trend is that the number of DB scans increases when *minsup* decreases. For GP-Close-*50000*, the tidsets can always fit in the tidset buffer after two DB scans. The number of DB scans performed by GP-Close-*0* increases rapidly as more branches of the enumeration tree are constructed when *minsup* decreases. However, for low *minsup*, though GP-Close-*0* and GP-Close-*500* scan for many more times than Cumulate, the performance of the two algorithms is still more than one order of magnitude better than Cumulate. This further reflects the fact that when *minsup* is low, the bottleneck of the algorithms lies in CPU computation instead of IO access.

Figure 3(c) and 3(f) show that the number of closed generalization closures is almost one to two orders of magnitude smaller than the number of all frequent

relationsets discovered by Cumulate. This is despite the fact that all frequent relationsets can be derived from the set of closed generalization closures. Note that a scale is used in Figure 3(c). Therefore, the stable margin between the two curves actually implies an exponential growth in the difference between the number of frequent generalized patterns and the number of closed closures.

6 Conclusion

This paper has presented an innovative approach for mining frequent generalized patterns from RDF metadata with *over-generalization reduction*. We presented the GP-Close algorithm which efficiently discovers a small set of closed frequent generalization closures from which all frequent generalized patterns can be derived. Extensive experiments show that our proposed method can substantially reduce the pattern redundancy and perform much better than the original GARM algorithm *Cumulate* in term of time efficiency.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: Semantic web. Scientific American **284**(5) (2001) 35–43
2. Srikant, R., Agrawal, R.: Mining generalized association rules. In: VLDB '95, San Francisco (1995) 407–419
3. Agrawal, R., Imielinski, T., Swami, A.N.: Mining association rules between sets of items in large databases. In: SIGMOD Conference. (1993) 207–216
4. Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining frequent patterns with counting inference. SIGKDD Explorations **2**(2) (2000) 66–75
5. Zaki, M.J., Hsiao, C.J.: Charm: An efficient algorithm for closed itemset mining. In: SDM. (2002)
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of VLDB'94, Santiago de Chile. (1994) 487–499
7. Hipp, J., Myka, A., Wirth, R., Güntzer, U.: A new algorithm for faster mining of generalized association rules. In: PKDD. (1998) 74–82
8. Sriphaew, K., Theeramunkong, T.: A new method for finding generalized frequent itemsets in generalized association rule mining. In: ISCC. (2002) 1040–1045
9. Inokuchi, A.: Mining generalized substructures from a set of labeled graphs. In: ICDM. (2004) 415–418
10. Ganter, B., Wille, R.: Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1997)