Journal of Web Engineering, Vol. 13, No.1&2 (2014) 097-113

A FAST APPROACH FOR QUERYING MULTIPLE ONTOLOGY VERSIONS BASED ON A CONCEPT LATTICE

YAQING LIU[1,2], RONG CHEN[1], YINGJIE SONG[1], WU DENG[1,3]

[1]*School of Information Science & Technology, Dalian Maritime University, Dalian, China*

[2]*Artificial Intelligence Key Laboratory of Sichuan Province, Sichuan University of Science and Engineering, Zigong, China*

[3]*Software Institute, Dalian Jiaotong University, Dalian, China*

liuyaqing234@yeah.net

Received March 16, 2013
Revised August 27, 2013

In this paper we propose a fast approach for querying multiple ontology versions, in which a novel model named as a version lattice based on a concept lattice is developed to serve as its foundation. We depict formally related problems of multiple ontology versions query and prove the equality between our description and a logic-based one. Supporting various requirements on multiple ontology versions, our approach can save more running time than previous algorithms, which is explained by analyzing our algorithms. Also experiments show that the advantage of running speed is more remarkable for ontology versions which are large in quantity and scale.

Key words: Ontology version, Version lattice, Concept lattice
Communicated by: G.-J. Houben & K. Turowski

1 Introduction

Ontology research has been increasing in popularity and ontologies plays an important role in many fields such as computer science, commerce, biology, etc, since the Semantic Web was firstly proposed in 2001[1]. Because the outer knowledge environment keeps changing constantly, ontologies need to be modified continuously. So a serious problem emerges: ontology evolution. Ontology evolution is the timely adaptation of an ontology to the arisen changes and the consistent propagation of these changes to dependent artifacts[2]. During ontology evolution, each modification of an ontology will generate a new version, which will cause many ontology versions to co-exist[13]. In order to manage ontology versions effectively, ontology versioning is proposed.

Ontology versioning is defined as "*the ability to handle changes in ontologies by creating and managing different variants of it*" [3]. For ontology developers, the latest ontology version is usually less stable than older versions because those ontology changes used to alter older versions have not yet been admitted by ontology users. Keeping multiple versions is good to withdraw or adjust the changes to avoid unintended results. For ontology users, they may prefer an earlier version because some

functions of earlier version are necessary to them. In addition, a lower requirement of resource for older version is also an important reason to select an older one. By now, a lot of research work[2-8] has been devoted to solving the above two problems. In view of the logical nature of ontologies, most of them are based on logic. The low query efficiency, which is very important, is neglected although the basic requirement of function is met. Specifically for large ontology versions both in quantity and in scale, the query time is unbearable[14]. In order to improve query efficiency, this paper proposes a novel approach based on a concept lattice for multiple ontology versions.

Unlike literatures [2] and [4], which only support queries between two ontology versions, our work is concerned with multiple ontology versions. Also, unlike literatures [5-8], which is focused on the quality of the query, our work is more concerned about the performance of query time. In current work we are trying to develop a new approach to querying multiple ontology versions. In this paper we report two contributions.

- We use formal concept analysis to build version lattice models for multiple ontology versions. The model is able to reveal clearly their similarities and differences.
- Seven algorithms are applied to multiple ontology versions query. We prove the correctness of these algorithms. The experiment shows that our algorithms can save more time than existing work.

This paper is organized as follow. Section 2 introduces basic definitions and conventions about concept lattices. Section 3 proposes the version lattice, a novel model for representing multiple versions. Section 4 shows how multiple version queries are solved. Section 5 shows the advantage of our approach by analyzing algorithms and executing experiments. Section 6 discusses related work. In the last section, we conclude the paper.

2 Formal Description of Concept Lattice

FCA(Formal Concept Analysis) is a very suitable tool for analyzing the relationship between objects and attributes. Concept lattice structure can be obtained when applying FCA(Formal Concept Analysis) to a formal context. Because multi-inheritance is supported, the concept lattice structure is more expressive than the tree model which only supports single-inheritance for concepts. FCA has been applied to many realms such as data mining, information extraction and program clustering[9].

2.1 Some Definitions of Concept Lattice

Definition 1. A **formal context** FT is defined as a 3-tuple: $FT=(B,A,P)$, where

- B is a group of formal objects.
- A is a group of formal attributes.
- $P \subseteq B \times A$. For any $b \in B$ and $a \in A$, $(b,a) \in P$ holds iff object b has attribute a.

Definition 2. Given a formal context $FT=(B,A,P)$ and $X \subseteq B$. $\delta(X)=\{a \in A | \forall b \in X: (b,a) \in P\}$ is said to be the **common attributes** of X.

Definition 3. Given a formal context $FT=(B,A,P)$ and $Y \subseteq A$. $\tau(Y)=\{b \in B | \forall a \in Y: (b,a) \in P\}$ is said to be the **common objects** of Y.

Definition 4. Given a formal context $FT=(B,A,P)$. A 2-tuple $fc=(X,Y)$ is said to be a **formal concept** of FT iff $Y=\delta(X)$ and $X=\tau(Y)$ hold, where $X\subseteq B$ and $Y\subseteq A$. X is said to be the **extent** of fc and Y is said to be the **intent** of fc.

Definition 5. Given a formal context $FT=(B,A,P)$. A **formal concept set** is defined as $FC=\{fc|fc$ is a formal concept of $FT\}$.

Definition 6. Given a formal context $FT=(B,A,P)$. $fc_1\leq fc_2$ holds iff $X_1\subseteq X_2$ and $Y_2\subseteq Y_1$ hold, where

- $fc_1, fc_2\in FC$ are two formal concepts of FT.
- fc_1, fc_2 are represented as $fc_1=(X_1,Y_1)$ and $fc_2=(X_2,Y_2)$, respectively.

Property 1. Given a formal concept set FC, $\leq$ is a partial order on FC.

Proof

$fc, fc_1, fc_2, fc_3\in FC$ are denoted as $fc=(X,Y)$, $fc_1=(X_1,Y_1)$, $fc_2=(X_2,Y_2)$ and $fc_3=(X_3,Y_3)$, respectively. The proof is shown as follow. Because $\leq$ is reflexive, anti-symmetric and transitive, $\leq$ is a partial order on FC.

$$\frac{\dfrac{X\subseteq X}{\forall fc\in FC,\ fc\leq fc}\quad \dfrac{X_1=X_2,\ Y_1=Y_2\rightarrow fc_1=fc_2}{X_1\subseteq X_2,\ X_2\subseteq X_1,\ Y_1\subseteq Y_2,\ Y_2\subseteq Y_1\rightarrow fc_1=fc_2}\quad \dfrac{X_1\subseteq X_3\rightarrow fc_1\leq fc_3}{X_1\subseteq X_2,\ X_2\subseteq X_3\rightarrow fc_1\leq fc_3}}{fc_1\leq fc_2,\ fc_2\leq fc_1\rightarrow fc_1=fc_2\quad fc_1\leq fc_2,\ fc_2\leq fc_3\rightarrow fc_1\leq fc_3}$$

$$\leq \text{ is a partial order on } FC$$

□

Definition 7. Given a formal context $FT=(B,A,P)$. $\underline{FC}=(FC,\leq)$ is said to be **concept lattice** of FT.

2.2 Representation of Concept Lattice

A concept lattice is usually represented as a line diagram because it can vividly represent the relationship between any two formal concepts. By now a lot of effective approaches have been proposed to generate line diagrams[11], so the algorithm of building line diagram is omitted and only related definitions and properties of it are shown as follow.

Definition 8. Given a concept lattice $\underline{FC}=(FC,\leq)$. fc_1 is said to **cover** fc_2 iff all of (1),(2) and (3) hold.

(1) $fc_1, fc_2\in FC$ and $fc_1\leq fc_2$ hold.
(2) $fc_1\neq fc_2$ holds.
(3) $fc_1\leq fc_2\wedge fc_2\leq fc_3\rightarrow fc_1=fc_2\vee fc_2=fc_3$ holds.

Definition 9. Given a formal context $FT=(B,A,P)$ and its concept lattice $\underline{FC}=(FC,\leq)$. $LG=(V,E)$ is **line diagram** of $\underline{FC}$, where

- There is a one-to-one correspondence between $v\in V$ and $fc\in FC$, which is represented as $oto(v,fc)$.
- $E\subseteq V\times V$ is the directed edge set of LG. And $\langle v_1,v_2\rangle\in E$ holds iff f_1 covers f_2, where $oto(v_1,fc_1)\wedge oto(v_2,fc_2)$ holds.

Definition 10. Given a line diagram $LG=(V,E)$. v_1 is said to be the **upper neighbour** of v_2 and v_2 is

said to be the **lower neighbour** of v_1 iff $<v_1,v_2>\in E$ holds.

Convention 1. Given a line diagram $LG=(V,E)$. $UNs(v)=\{v'|<v,v'>\in E\}$ is set of all upper neighbours. $LNs(v)=\{v'|<v',v>\in E\}$ is set of all lower neighbours.

Definition 11. Given a line diagram $LG=(V,E)$. For $\forall v\in V$, the **name of *v*** is defined as $(X_u\text{-}X,Y\text{-}Y_l)$, where $oto(v,(X,Y))\wedge oto(v_u,(X_u,Y_u))\wedge oto(v_l,(X_l,Y_l))$ holds, where $<v_u,v>\in E$ and $<v,v_l>\in E$.

Definition 12. Given a line diagram $LG=(V,E)$, v_1 is said to be the **descendant** of v_2 and v_2 is said to be the **ancestor** of v_1 iff $<v_1,v_2>\in E$ holds or $\exists v'_1,v'_2,\ldots,v'_n\in V$, $<v_1,v_1'>\in E\wedge<v_1',v_2'>\in E\wedge\ldots\wedge<v_n',v_2>\in E$ holds.

Convention 2. Given a line diagram $LG=(V,E)$. $DEs(v)=\{v'|\ v'$ is a descendant of $v\}$ is set of all descendants. $ANs(v)=\{v'|\ v'$ is an ancestor of $v\}$ is set of all ancestors.

Convention 3. Given a line diagram LG. We use signs $V(LG)$ and $E(LG)$ to represent the vertex set and the edge set of LG, respectively.

Convention 4. Given a line diagram $LG=(V,E)$ and a vertex v, v is represented as $\uparrow$ iff $\neg\exists v'\in V$, $<v,v'>\in E$ holds.

Convention 5. Given a line diagram $LG=(V,E)$ and a vertex v, v is represented as $\downarrow$ iff $\neg\exists v'\in V$, $<v',v>\in E$ holds.

Property 2. Given a line diagram $LG=(V,E)$ and a vertex $v=(X,Y)$. For $\forall x\in X$ and $\forall y\in Y$, x has y.

Proof

By definition 9, there is a one-to-one correspondence between an vertex $v=(X_v,Y_v)$ and a formal concept $(X,Y)\in FC$. By definition 4, $\forall x\in X$ and $\forall y\in Y$, formal object x has formal attribute y. By definition 11, $X_v\subseteq X\wedge Y_v\subseteq Y$ holds. So, $\forall x\in X_v$ and $\forall y\in Y_v$, formal object x has formal attribute y.

□

Property 3. Given a line diagram $LG=(V,E)$. $\forall x_1\in X_1$ and $\forall y_2\in Y_2$, x_1 has y_2 if (X_1,Y_1) is a lower neighbour of (X_2,Y_2), where $(X_1,Y_1),(X_2,Y_2)\in V$.

Proof

The proof is shown as follow.

$$\dfrac{\dfrac{\dfrac{\dfrac{(X_1,Y_1)\text{ is a lower neighbour of }(X_2,Y_2)}{(X_1,Y_1)\text{ covers }(X_2,Y_2)}}{(X_1,Y_1)\leq(X_2,Y_2)}}{\dfrac{y_2\in Y_2 \qquad Y_2\subseteq Y_1}{x_1\in X_1 \qquad y_2\in Y_1 \qquad (X_1,Y_1)\text{ is a formal concept}}}}{x_1\text{ has }y_2}$$

□

Property 4. Given a line diagram $LG=(V,E)$. $\forall x_1\in X_1$ and $y_2\in Y_2$, x_1 has y_2 if (X_1,Y_1) is a descendant of (X_2,Y_2), where $(X_1,Y_1),(X_2,Y_2)\in V$.

Proof

The proof is shown as follow.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(X_1,Y_1) \text{ is a descendant of } (X_2,Y_2)}{<(X_1,Y_1),(X_1',Y_1')>\in E\wedge<(X_1',Y_1'),(X_2',Y_2')>\in E\wedge\ldots\wedge<(X_n',Y_n'),(X_2,Y_2)>\in E}}{(X_1,Y_1) \text{ covers } (X_1',Y_1')\wedge(X_1',Y_1') \text{ covers } (X_2',Y_2')\wedge\ldots\wedge(X_n',Y_n') \text{ covers } (X_2,Y_2)}}{(X_1,Y_1)\leq(X_1',Y_1')\wedge(X_1',Y_1')\leq(X_2',Y_2')\wedge\ldots\wedge(X_n',Y_n')\leq(X_2,Y_2)}}{\exists(X_1',Y_1'),(X_2',Y_2'),\ldots,(X_N',Y_N'),\ Y_2\subseteq Y_1'\wedge Y_1'\subseteq Y_2'\wedge\ldots\wedge\ Y_n'\subseteq Y_1}$$

$$\dfrac{\dfrac{y_2\in Y_2 \qquad Y_2\subseteq Y_1}{x_1\in X_1 \qquad y_2\in Y_1} \qquad (X_1,Y_1) \text{ is a formal concept}}{x_1 \text{ has } y_2}$$

□

3 Formal Description of Version Lattice

3.1. Ontology and Ontology version

Definition 13. An **ontology** O is defined as a 5-tuple[10]: $O=\{C,R,H^c,Rel,A^o\}$
where:

- C is the set of classes, which represents various entities in some domain being modeled. We assume that classes are named by one or more natural language terms and are normally referenced within the ontology by a unique identifier.
- $H^C\subseteq C\times C$ is a set of taxonomic relationships between classes. Such relations define the hierarchy of classes.
- R is the set of non-taxonomic relationships. The function Rel: $R\rightarrow C\times C$ maps the relation identifiers to the actual relationships.
- A^o is a set of axioms, usually formalized into some logic language. These axioms specify additional constraints on the ontology and can be used in ontology consistency checking and for inferring new knowledge from the ontology through some inference mechanism.

For brevity, O is short for ontology throughout this paper. Its class set is denoted by $O.C$, taxonomic relationships by $O.H^C$.

Definition 14. $Os=\{O_1,O_2,\ldots,O_n\}$ is said to be the **ontology space** of O iff $\forall 1\leq i\leq n$, O_i is a version of ontology O.

For example, the ontology space $Os=\{O_1,O_2,O_3,O_4,O_5\}$ about ontology O is shown in Figure 1. Each class is denoted by an ellipse. All of c_1, c_2, c_3, c_4 are classes of O. Every taxonomic relationship is denoted by an arrow from subclass to superclass. An example is that c_1 is a subclass of c_2 in O_1

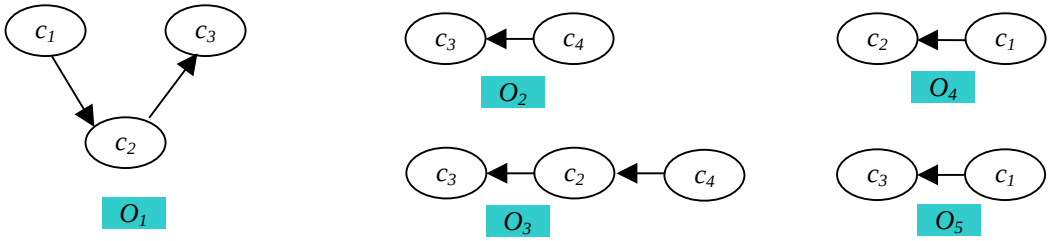

Figure 1 An example ontology space $Os=\{O_1, O_2, O_3, O_4, O_5\}$

3.2. Version Lattice

By the taxonomic relationship of an ontology, a formal context for its ontology space can be built.

Definition 15. Given an ontology space $Os=\{O_1,O_2,\ldots,O_n\}$, $FT_{Os}=(B,A,P)$ is said to be a **version formal context**, where

- $B:=Os$.
- $A:=O_1.H^C\cup O_2.H^C\cup\ldots\cup O_n.H^C$.
- For any $b\in B$ and $a\in A$, $(b,a)\in P$ holds iff $a\in b.H^C$ holds.

For example, the version formal context of the ontology space $Os=\{O_1, O_2, O_3, O_4, O_5\}$ shown in Figure 1 is shown in Figure 2. For any $(c_i,c_j)\in A$, c_i is subclass of c_j. if $(b,a)\in P$ holds, the character '×' appears across b and a.

Convention 5. Given a version formal context $FT_{Os}=(B,A,P)$. We use $\underline{FC_{Os}}=(FC_{Os},\leq)$ to represent the concept lattice of FT_{Os} (named as the version lattice) and use $LG_{Os}=(V,E)$ to represent the line diagram of $\underline{FC_{Os}}$.

The line diagram of the version lattice based on the version formal context shown in Figure 2 is shown in Figure 3.

	(c_1,c_2)	(c_4,c_3)	(c_2,c_3)	(c_1,c_3)	(c_4,c_2)
O_1	×		×		
O_2		×			
O_3			×		×
O_4	×				
O_5				×	

Figure 2 The version formal context based on the ontology space shown in Figure 1

4 Query Algorithms for Class Hierarchy

In paper [8], seven cases of multiple ontology versions queries for the class hierarchy of ontology versions were proposed. In our paper, these seven cases may be divided into two types. One is named as multiple ontology versions query without parameters and the other is named as multiple ontology versions query with parameters.

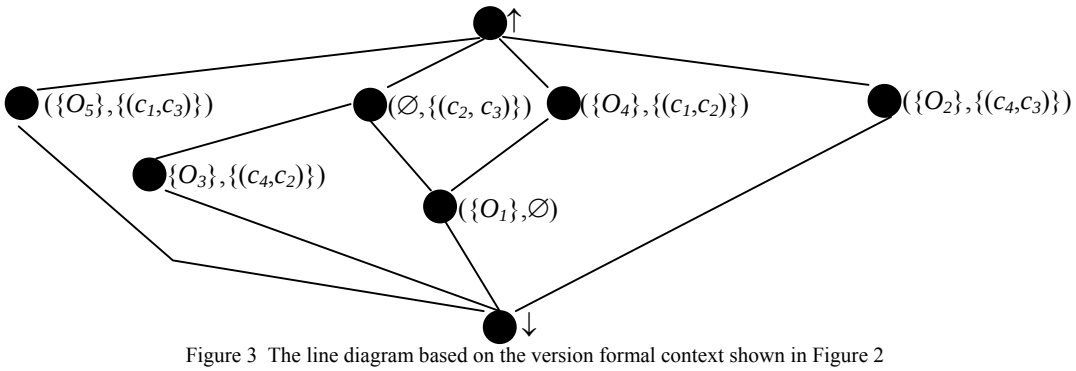

Figure 3 The line diagram based on the version formal context shown in Figure 2

The performance of query is enhanced greatly because the search space is decreased greatly in algorithms 1-7. In algorithms 1-7, the traversed objects are vertices rather than ontology versions themselves. And similarities and differences between ontology versions about the class hierarchy are displayed in vertices. Because these queries are to find similarities and differences between ontology versions, only a small subset of V is checked. These results of queries such as 1, 2 and 3 can be provided directly by the line diagram.

4.1. Query Algorithms for Multiple Ontology Versions without Parameters

$Os=\{O_1,O_2,\ldots,O_n\}$ denotes an ontology space throughout this paper.

Query 1: Search all taxonomic relationships which always hold in every ontology version.

Those all taxonomic relationships which always hold in every ontology version can be described formally as $O_1.H^C \cap O_2.H^C \cap \ldots \cap O_n.H^C$.

Lemma 1. $O_1.H^C \cap O_2.H^C \cap \ldots \cap O_n.H^C = \delta(Os)$ holds

Proof

$\delta(Os)$ is the set of common attributes of all ontology versions. Vertex$\uparrow$= $(O_s,\delta(Os))$ is the top most in LG_{OS}. For $\forall O \in O_s$ and $\forall h \in \delta(Os)$, h holds in O by property 2. So $O_1.H^C \cap O_2.H^C \cap \ldots \cap O_n.H^C = \delta(Os)$ holds.

□

By lemma 1, algorithm 1 is used to find all taxonomic relationships which always hold in every ontology version.

Algorithm 1: Diff(Os)
Input: Os, ontology space
Output: COMMON, all taxonomic relationships which always hold in Os
1. COMMON:=∅;
2. (X,Y):=*getTopVex*(LG_{Os}); //Function *getTopVex*(LG_{Os}) is used to get the vertex ↑ from LG_{Os}
3. COMMON:=Y;
4. RETURN COMMON

Query 2: Given an ontology version $O_i \in Os(1 \le i \le n)$, search those taxonomic relationships which hold in O_i and don't hold in $Os-\{O_i\}$.

Those taxonomic relationships which hold in O_i and don't hold in $Os-\{O_i\}$ can be described formally as $New=O_i.H^C-O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C$.

Lemma 2. $New=Y$ holds if $\{O_i\}=X$ holds, where $\downarrow=(X,Y)$.

Proof

Step 1 for lemma 2

$$\frac{\dfrac{\{O_i\}=X}{(Y \subseteq O_i.H^C)} \quad \dfrac{\downarrow=(X,Y)}{Y \cap (O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C)=\varnothing}}{\dfrac{Y \subseteq O_i.H^C-O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C}{Y \subseteq New}}$$

Step 2 for lemma 2

$$\frac{\dfrac{\dfrac{\{O_i\}=X \qquad \downarrow=(X,Y)}{Y \cup O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C=O_1.H^C \cup \ldots \cup O_n.H^C}}{O_i.H^C-O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C \subseteq Y}}{New \subseteq Y}$$

We prove $Y \subseteq New$ in step 1 and prove $New \subseteq Y$ in step 2. According to step1 and step 2, $New=Y$ is proved.

□

Lemma 3. $New=Y$ holds if $\{O_i\}=X$ holds and $\forall (X',Y') \in DEs((X,Y))$, $X'=\varnothing$ hold, where $(X,Y) \in V(LG_{Os})$ and $(X,Y) \neq \downarrow$.

Proof

We prove $Y \subseteq New$ in step 1 and prove $New \subseteq Y$ in step 2. According to step1 and step 2, $New=Y$ is proved.

Step 1for lemma 3

$$\frac{\dfrac{\{O_i\}=X}{(Y \subseteq O_i.H^C)} \quad \dfrac{\forall (X',Y') \in DEs((X,Y)),\ X'=\varnothing}{Y \cap (O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C)=\varnothing}}{\dfrac{Y \subseteq O_i.H^C-O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C}{Y \subseteq New}}$$

Step 2 for lemma 3

$$\frac{\dfrac{\dfrac{\{O_i\}=X \qquad \forall (X',Y') \in DEs((X,Y)),\ X'=\varnothing}{Y \cup O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C=O_1.H^C \cup \ldots \cup O_n.H^C}}{O_i.H^C-O_1.H^C \cup O_2.H^C \cup \ldots \cup O_{i-1}.H^C \cup O_{i+1}.H^C \cup \ldots \cup O_n.H^C \subseteq Y}}{New \subseteq Y}$$

□

By lemma 2 and lemma 3, algorithm 2 is used to search those taxonomic relationships which hold in a special ontology version O and don't hold in Os-$\{O\}$.

Algorithm 2: NewTaxRel(Os,O)
Input: Os, ontology space and $O \in Os$, a special ontology version
Output: New, taxonomic relationships which hold in O and don't hold in Os-$\{O\}$.

```
1.  New←∅;
2.  // Lines 3-7 are responsible to find the vertex which includes O.
3.  WHILE(hasMoreVertices(LG_Os,↓)≠∅)
4.  //Function hasMoreVertices (LG_Os,↓) is used to traverse all vertices from ↓ to ↑ in LG_Os
5.    (X,Y):=getNextVertex(LG_Os);//Function getNextVertex(LG_Os) is used to get a vertex.
6.    IF O∈X
7.      BREAK;
8.  IF ↓=(X,Y) OR ∀(X',Y')∈Des((X,Y)), X'=∅ //By lemma 2, 3, Y is assigned to New, or not.
9.    New:=Y;
10. RETURN New
```

Query 3: Given an ontology version $O_i \in Os$, search those taxonomic relationships which don't hold in O_i and hold in Os-$\{O_i\}$.

Those taxonomic relationships which don't hold in O_i and hold in Os-$\{O_i\}$ can be described formally as $Del=O_1.H^C \cap O_2.H^C \cap \ldots \cap O_{i-1} \cap O_{i+1} \cap \ldots \cap O_n.H^C - O_i.H^C$.

Lemma 4. $Del = Y_1$ holds if all of (1), (2) and (3) hold.

(1) $\{O_i\}=X$

(2) $\uparrow=(X,Y)$

(3) $(X_1,Y_1) \in LN_s((X,Y))$ and $|LN_s((X,Y))|=1$.

Proof

$$\dfrac{\dfrac{\uparrow=(X,Y)}{O_1.H^C \cap \ldots \cap O_n.H^C=Y} \qquad \dfrac{\uparrow=(X,Y) \quad \{O_i\}=X \quad (X_1,Y_1) \in LN_s((X,Y)) \quad |LN_s((X,Y))|=1}{O_1.H^C \cap O_2.H^C \cap \ldots \cap O_{i-1} \cap O_{i+1} \cap \ldots \cap O_n.H^C=Y \cup Y_1}}{\dfrac{O_1.H^C \cap O_2.H^C \cap \ldots \cap O_{i-1} \cap O_{i+1} \cap \ldots \cap O_n.H^C - O_i.H^C=Y_1}{Del=Y_1}}$$

□

By lemma 4, algorithm 3 is used to search these taxonomic relationships which don't hold in a selected ontology version O and hold in Os-$\{O\}$.

Algorithm 3: DelTaxRel(*Os*,*O*)
Input: *Os*, ontology space and $O \in Os$, a special ontology version
Output: `Del`, taxonomic relationships which don't hold in *O* and hold in *Os*-{*O*}.
1. `Del`:=∅;
2. (*X*,*Y*):=*getTopVertex*(LG_{Os});
3. //Function *getTopVertex*(LG_{Os}) is used to get vertex ↑.
4. IF {*O*}=*X* AND (X_1,Y_1)∈LN_s((*X*,*Y*)) AND |LN_s((*X*,*Y*))|=1
5. //By lemma 4, Y_1 is assigned to *New*, or not.
6. `Del`:=Y_1;
7. RETURN `Del`

4.2. Query Algorithms with Parameters

Query 4: Given a taxonomic relationship $h \in O_1.H^C \cup O_2.H^C \cup \ldots \cup O_n.H^C$, search for ontology version O_m where O_m has *h* and the maximum index, $h \in O_m.H^C \wedge \forall m<i \leq n, h \notin O_i.H^C$.

Algorithm 4 is used to search ontology version O_m.

Algorithm 4: NewVersion(*Os*,*h*)
Input: *Os*, ontology space and *h*, a selected taxonomic relationship
Output: `O`, an ontology version
1. *Candidate*:=∅; //∀*O*∈*Candidate*, *O* has *h*.
2. //Lines 3-8 are responsible to find the first ontology version which has *h*.
3. WHILE(*hasMoreVertices*(LG_{Os},↑)≠∅)
4. //Function *hasMoreVertices*(LG_{Os},↑) is used to traverse all vertices from ↑ to ↓ in LG_{Os}
5. (*X*,*Y*):=*getNextVertex*(LG_{Os});//Function *getNextVertex*(LG_{Os}) is used to get a vertex.
6. IF *h*∈*Y*
7. *Candidate*:=*Candidate*∪*X*;
8. BREAK;
9. //Lines 10-13 is responsible to find all ontology versions any of which has *h*.
10. WHILE(*hasMoreVerticex*(LG_{Os},*DEs*((*X*,*Y*)))≠∅)
11. //Function *hasMoreVerticex*(LG_{Os},*DEs*((*X*,*Y*))) is used to traverse all vertices from (*X*,*Y*) to ↓
12. (*X'*,*Y'*):=*getOneVertex*(LG_{Os}); //Function *getNextVertex*(LG_{Os}) is used to get a vertex.
13. *Candidate*:=*Candidate*∪*X'*;// For ∀*O*∈*X'*, *O* has *h* by property 4.
14. `O`:=*Max*(*Candidate*);
15. //Function *Max*(*Candidate*) is used to find the ontology version which has max index in *Candidate*.
16. RETURN `O`

Query 5: Given an ontology class $c \in O_1.C \cup O_2.C \cup \ldots \cup O_n.C$, search for ontology classes set *commonChildren*$=\{c' | (c,c') \in O_1.H^C \cap O_2.H^C \cap \ldots \cap O_n.H^C\}$.

Algorithm 5 is used to search for ontology version ontology classes set *commonChildren*.

Algorithm 5: getCommonChildren(*Os*,*c*)
Input: *Os*, ontology space and *c*, an ontology class
Output: `Children`, ontology classes set
1. `Children`:=∅;
2. (*X*,*Y*):=↑;
3. //Lines 4-6 is explained by convention 5.
4. FOR EACH (c_1,c_2) in *Y*
5. IF $c==c_1$
6. `Children`:=`Children`∪{c_2};
7. RETURN `Children`

Query 6: Given an ontology class $c\in O_1.C\cup O_2.C\cup\ldots\cup O_n.C$ and an ontology version $O_i\in Os$, search for ontology classes set $newChildren=\{c'|(c,c')\in O_i.H^C\wedge(c,c')\notin O_1.H^C\cup O_2.H^C\cup\ldots\cup O_{i-1}.H^C\cup O_{i+1}.H^C\cup\ldots\cup O_n.H^C\}$.

Algorithm 6 is used to search for ontology version ontology classes set *newChildren*.

Algorithm 6: getNewChildren(*Os*,*O*,*c*)
Input: *Os*, ontology space and *O*, a special ontology version and *c*, an ontology class
Output: `Children`, ontology classes set
1. `Children`:=∅;
2. //Lines 3-7 is responsible to find the vertex which has *O*.
3. WHILE(*hasMoreVertices*(LG_{Os},↓)≠∅)
4. //Function *hasMoreVertices*(LG_{Os},↓) is used to traverse all vertices from ↓ to ↑ in LG_{Os}.
5. (*X*,*Y*):=*getNextVertex*(LG_{Os}); //Function *getNextVertex*(LG_{Os}) is used to get a vertex.
6. IF $O\in X$
7. BREAK;
8. IF (*X*,*Y*)=↓ OR $\forall(X_1,Y_1)\in DEs((X,Y))$, $X_1=\emptyset$
9. //Line 8 is explained by lemma 2 and 3, c_2 is objective ontology class.
10. FOR EACH (c_1,c_2) in *Y*
11. IF $c==c_1$
12. `Children`:=`Children`∪{c_2};
13. RETURN `Children`

Query 7: Given an ontology class $c\in O_1.C\cup O_2.C\cup\ldots\cup O_n.C$ and an ontology version $O_i\in Os$, search for ontology classes set $loseChildren=\{c'|(c,c')\notin O_i.H^C\wedge(c,c')\in O_1.H^C\cap O_2.H^C\cap\ldots\cap O_{i-1}.H^C\cap O_{i+1}.H^C\cap\ldots\cap O_n.H^C\}$.

Algorithm 7 is used to search for ontology version ontology classes set *loseChildren*.

Algorithm 7: getLoseChildren(*Os*,*O*,*c*)
Input: *Os*, ontology space and *O*, a special ontology version and *c*, an ontology class
Output: `Children`, ontology classes set
1. `Children`:=∅;
2. (*X*,*Y*):= ↑;
3. IF *O*={*X*} AND |*LNs*((*X*,*Y*))|==1 AND (*X*$_1$,*Y*$_1$)∈*LNs*((*X*,*Y*))
4. //Line 3 is explained by lemma 4. *c*$_2$ is objective ontology class.
5. For EACH (*c*$_1$,*c*$_2$) in *Y*$_1$
6. IF *c*==*c*$_1$
7. `Children`:=`Children`∪{*c*$_2$};
8. RETURN `Children`

5 Analysis and Experiments of Algorithms

5.1. Complexity Analysis for Our Algorithms

In paper [8], a prototypical system named as MORE, which is based on a description logic reasoner, was implemented for multiple version management. As is well known, the core algorithm for description logic reasoners is Tableau, which has exponential time complexity in the worst case. Compared with Tableau, the time complexity of our approach is in the worst case polynomial.

We stated the best time complexity and the worst one of our approach in Table 1. The complexity results of algorithms 1-7 are explained as follow.

- For algorithm 1, the output "*common*" is got directly from vertex ↑. So the complexity is always *O(1)* both in best situations and worst situations.
- For algorithm 2, lines 3-7, a loop sentence, is responsible to find the vertex which includes *O*. Function *hasMoreVertices*(LG_{Os}, ↓) is used to traverse all vertices from ↓ to ↑ in LG_{Os}. If ↓=(*X*,*Y*) holds, the best time complexity is *O(1)*. The worst situation is that both ↑=(*X*,*Y*) and ∀(*X'*,*Y'*)∈*Des*((*X*,*Y*)), *X'*=∅ hold. So, the worst time complexity is *O(n)*, where *n* is the number of vertices in LG_{Os}.
- For algorithm 3, the output "*common*" is got from vertex ↑ and its neighbour. So the complexity is always *O(1)* both in best situations and worst situations.
- For algorithm 4, lines 3-8, a loop sentence, is responsible to find the first ontology version which has *h*. Function *hasMoreVertices*(LG_{Os},↑) is used to traverse all vertices from ↑ to ↓ in LG_{Os}. If both (*X*,*Y*)=↑ and *h*∈*Y* hold, the best time complexity is *O(1)*. If both (*X*,*Y*)=↓ and *h*∈*Y* hold, the worst time complexity is *O(n)*, where *n* is the number of vertices in LG_{Os}. Lines 10-13 is responsible to find all ontology versions any of which has *h*. Function *hasMoreVerticex*(LG_{Os},*DEs*((*X*,*Y*))) is used to traverse all vertices from (*X*,*Y*) to ↓. If both (*X*,*Y*)=↓ and *h*∈*Y* hold, the best time complexity is *O(1)*. If both (*X*,*Y*)=↑ and *h*∈*Y* hold, the worst time complexity is *O(k)*, where *k* is cardinality of *DEs*((*X*,*Y*)). Function *Max*(*Candidate*) is used to find the ontology version which has max index in *Candidate*. If

both $(X,Y)=\downarrow$ and $h \in Y$ hold, the best time complexity is *O(1)*. If both $(X,Y)=\uparrow$ and $h \in Y$ hold, the worst time complexity is *O(k)*.

- For algorithm 5, lines 4-6, a loop, is responsible to traverse all elements in *Y*. So, the time complexity is *O(k)*, where *k* is the cardinality of *Y*.
- For algorithm 6, lines 3-7, a loop, is responsible to find the vertex which has *O*. Function *hasMoreVertices*(↓) is used to traverse all vertices from ↓ to ↑ in LG_{Os}. If both $(X,Y)=\downarrow$ and $O \in X$ hold, the best time complexity is *O(1)*. If both $(X,Y)=\uparrow$ and $O \in X$ hold, the worst time complexity is *O(n)*, where *n* is the number of vertices in LG_{Os}. Lines 8-12, a loop, are responsible to traverse all elements in *Y*. If $(X,Y)=\downarrow$ holds, the time complexity is *O(k)*, where *k* is the cardinality of *Y*. If $(X,Y)\neq\downarrow$ holds, the time complexity is 1.
- For algorithm 7, lines 5-7, a loop, is responsible to traverse all elements in Y_1. The time complexity is *O(k)*, where *k* is the cardinality of Y_1.

Table 1 Time complexity of our approach

Query times	A1	A2	A3	A4	A5	A6	A7
Best Time complexity	*O(1)*	*O(1)*	*O(1)*	*O(k)+O(k), 2k<n* *O(n), 2k>n*	*O(k)*	*O(k)*	*O(k)*
Worest Time complexity	*O(1)*	*O(n)*	*O(1)*	*O(k)+O(k), 2k>n* *O(n), 2k<n*	*O(k)*	*O(n)+O(k)*	*O(k)*

In Table 1, the first line denotes algorithms 1-7. The other lines claim the time complexity of the corresponding algorithm.

5.2. Experiment and *Results*

The experimental environment is a 16G RAM, Intel P3770 3.9GHZ, Windows XP OS. We compare our approach with a Tableau algorithm. In our experiment, the reasoner Pellet 2.2.0, which is based on the Tableau algorithm, is used. Pellet is open source tool and implemented by Maryland university. Pellet can be downloaded from http://clarkparsia.com/pellet.

(1) Experiment for an education ontology *E*

Table 2 the number of taxonomic relationships for each ontology version

Versions of *E*	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}
Number of taxonomic relationship	510	532	524	558	568	601	615	628	624	641

Table 3 the running time of our approach & Tableau for *E* (unit: ms)

Name	**Number of versions**	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**
Our approach	5	0	0	0	47	16	15	15
Tableau	5	360	371	377	38	355	305	296
Our approach	8	0	0	0	94	16	15	15
Tableau	8	640	595	625	48	611	588	612
Our approach	10	0	0	0	125	16	15	15
Tableau	10	812	822	799	35	773	791	802

E has ten versions E_1, E_2, …, E_{10}. The number of taxonomic relationship of every version is shown in Table 2. We compare our approach with Tableau on the time complexity of each query in Table 3. Three tries were done in total in our experiment. Three ontology versions, five ontology versions and ten ontology versions are set as ontology space, respectively. The results of the experiment are shown in Figure 4. All of the running time of our approach for algorithms *A1*, *A2*, *A3* are 0, which is explained that the time was too low to measure.

Figure 4 The running time about Tableau & our approach

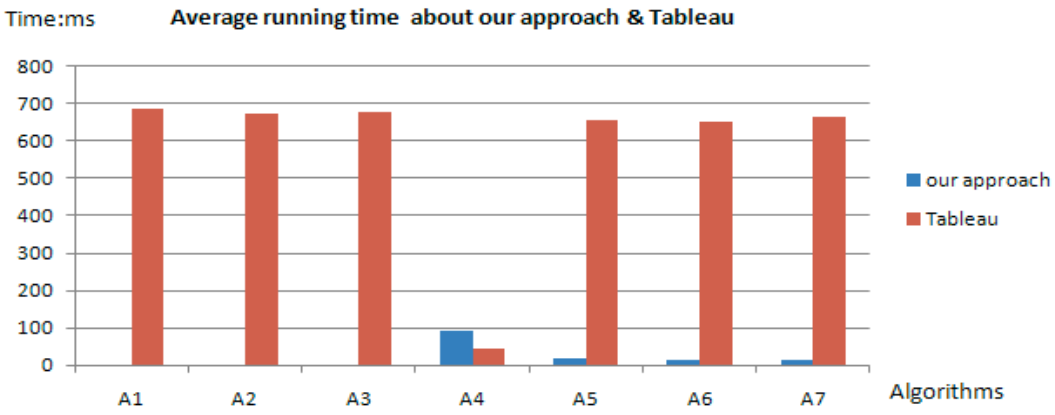

Figure 5 The average running time about Tableau & our approach

(2) Experiment for ontology OJKL and BioSAIL

We have compared our approach with Tableau on different versions of real life ontologies from different domains. In the following, we briefly report experiments we performed on detecting changes in the class hierarchy of the following two ontologies.

The OJKL ontology (Ontology of the Juridical Legal Knowledge) is a legal Ontology that has been developed in the SEKT project [15]. We used five different versions of the ontology from different stages of the development process. Each of these versions contains about 80 classes. The BioSAIL ontology was developed within the BioSTORM project[16]. We take three versions of the BioSAIL ontology for the tests reported below. Each version of BioSAIL ontology has about 180 classes.

The result is stated in Table 4.

Table 4 the running time of our approach & Tableau for OJKL and BioSAIL (unit: ms)

Name	**Ontology**	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**
Our approach	OJKL	0	0	0	13	8	7	7
Tableau		85	87	88	10	83	81	81
Our approach	BioSAIL	0	0	0	25	8	8	6
Tableau		177	181	183	22	170	163	163

(3)Comments

For Tableau, the running time is increasing by a wide margin as the number of versions increasing, which is shown by Figure 4. In contrast to the running time of Tableau, the running time of our approach is almost unchanged as the number of versions increases. According to Figure 5, the average running time of our approach is much less than Tableau.

By Table 3 and Table 4, algorithms 1, 2, 3, 5, 6 and 7 beat Tableau in performance of query. But algorithm 4 is a little weaker than Tableau. By analyzing the three ontologies, we find the vertex containing taxonomic relationship h is very near to vertex $\downarrow$. So in our experiments, the time complexity of algorithm 4 is almost worst because the time complexity of algorithm 4 is proportional to the number of vertices in line diagram. But for query 4, Tableau traverses these ontology versions from the max indexed ontology to the min indexed one. Most of ontology versions are not traversed because the h is contained in the greater indexed ontology and is found earlier. It is faster to find answer because there is less deduction in query 4 than other queries. This is why the switches take place in performance for A4.

6 Related Work

A lot of research work has been done on ontology versioning[12]. The research approaches are mainly based on logic.

Inspired by these similar problems, which software engineering have faced for many years, Noy proposes an ontology management framework, PROMPT, for comparing different ontologies on structure and mapping similarities and differences between them[2]. Given any two ontologies, three result items, "*renamed*", "*operation*" and "*map level*" are shown by PROMPTDIFF. According to a set of heuristic matchers, PROMPT is able to map between any two ontologies. A bi-directional map between two ontologies BOV is presented[4]. Four mapping relations, "*Equivalence*", "*Isomorphic*",

"*Isomerours*" and "*Mutiple_Change*", are defined to describe the relationship between mapping elements. Linguistic similarity and structural similarity are computed according to mapping relations between two ontologies. A threshold th_{accept} is set to identify the mappings. Compared to related approaches, the major advantage of BOV is that less time and effort are required for mapping identification and transformation in bi-directions. But PROMPT and BOV are only adaptive to two ontologies and they do not provide support for query across multiple ontologies. If the approaches in PROMPT or BOV are used to query across multiple ontologies, it will become a NP problem.

Based on the logical nature of ontologies, logic approaches are proposed to highlight differences between multiple ontologies and support queries on them. Theoretically, Leenhee presents a model-independent ontology evolution framework for revising and managing multiple ontology versions in a possible worlds setting[5]. Based on AGM theory, Leenhee defines three different transformation types for depicting ontology evolution: expansion transformation, revision transformation and contraction transformation. In addition, equivalence-preserving, possibility and necessity of ontology evolution are discussed at length. Furthermore, Heflin gives a formal description and account for three kinds of links between semantic web documents: commitment to an ontology by a resource, extension of one ontology by another, and backward-compatibility of an ontology with a prior version[6]. Practically, OntoView, a tool supporting ontology versioning, is developed[7]. For two ontologies on the web, any of non-logical change, logical definition change, identifier change, addition of definitions and deletion of definitions is shown to users by OntoView. Huang extends temporal logic by adding three temporal operators: previous operator, sometimes-in-the-past operator and always-in-the-past operator[8]. In addition LTM, a query language based on temporal logic is defined to provide reasoning queries and retrieval queries to users.

Logic approaches, such as these, are more expressive for highlighting differences between them and supporting queries to them. But when facing large ontologies both in quantity and in scale, the running efficiency of multiple versions query decreases heavily. Our approach is able to provide all 7 queries proposed in [8], but the running efficiency of our approach is higher.

7 Conclusions and Future Work

In this paper, we discussed the running efficiency of querying multiple ontology versions. We proposed a novel management model based on a concept lattice for representing the relationship among multiple ontology versions, and then mathematically described the query requirement on multiple ontology versions. Based on a version lattice, we showed higher running efficiency by theory and experiments.

Next, we intend to look at differences between classes, class properties, axioms, etc. The approach based on concept lattices should be applied to them easily. One example is that individuals and classes are considered as formal attributes and formal objects, respectively. Another is that properties and classes are viewed as formal attributes and formal objects, respectively. Perhaps, the difference of axioms is more difficult because these axioms, which are represented by description logic language generally, are hard to be expressed as formal attributes. A potential approach to query cross multiple versions with axioms is based on decomposition of definition and structure of axioms.

Acknowledgements

This work was supported by National Natural Science Foundation of China (61175056, 61203283), Young Key Teachers Foundation Projects of Dalian Maritime University (2012QN031), the Open Project Program of Artificial Intelligence Key Laboratory of Sichuan Province (2013RYJ02, 2012RYJ02), the Higher growth plans of Liaoning Province for Distinguished Young Scholars (LJQ2013049)，State Key Laboratory of Software Engineering (SKLSE) (SKLSE2012-09-27), the Open Project Program of Sichuan Provincial Key Lab of Process Equipment and Control (GK201202), the Open Project Program of the Traction Power State Key Laboratory of Southwest Jiaotong University (TPL1203), the Open Project Program of Guangxi Key Laboratory of Hybrid Computation and IC Design Analysis (Guangxi University for Nationalities) (2012HCIC06).

References

1. Berners-Lee, T., Hendler J. and Lassila O., The Semantic Web. Scientific American, 2001.
2. Noy, N.F., Musen M.A., Ontology Versioning in an Ontology Management Framework. IEEE Intelligent Systems, 19(4):6-13(2004).
3. Klein, M., and Fensel, D., Ontology versioning for the Semantic Web. In SWWS'2001, (2001).
4. Zhao,S. and Tierney B., Bi-directional Ontology Versiong BOV. Lecture Notes in Computer Science, 2005, 906-912.
5. Leenhee, P.D., Revising and Managing Multiple Ontology Versions in a Possible Worlds Setting. Lecture Notes in Computer Science, 2004, 798-809.
6. Heflin, J and Pan, Z., A Model Theoretic Semantics for Ontology Versioning. In ISWC'2004, (2004).
7. Klein, M., Fensel, D., Kiryakov, A., Ognyanov, D., Ontology versioning and change detection on the Web. Lecture Notes in Computer Science, 2002, 247-259.
8. Huang, Z. and Stuckenschmidt, H., Reasoning with Multi-version Ontologies: A Temporal Logic Approach. Lecture Notes in Computer Science, 2005, 398-412.
9. Tilley, Thomas., Cole, R., Becker, P and Eklund, P., A Survey of Formal Concept Analysis Support for Software Engineering Activities. Lecture Notes in Computer Science, 2005, 250-271.
10. Stojanovic, L. Methods and Tools for Ontology Evolution. PhD thesis, University of Karlsruhe, Germany, 2004.
11. Godin, R., Missaoui, R and Alaoui, H., Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence, 11(2):246-267 (1995).
12. Liu, Y., Chen, R and Du, Z., A novel model for ontology versions maintenance. Journal of Computational Information Systems, 7(14):5201-5209, 2011.
13. Liu, Y., Chen, R., Gao, J and Yang, H., A Conflict-Resolving Approach to Ontology Evolution in Open Environments. Engineering Intelligent System, 18(3/4): 223-231, 2010.
14. Li, Y., Comparison and Analysis of Programming Ontology Reasoner. Journal of South China Normal University, 44(3):59-63, 2011.
15. Casanovas, Pompeu; Poblet, Marta; Casellas, Nuria; Contreras, Jess; Benjamins, V.Richard; Blzquez, Mercedes., Supporting newly-appointed judges: A legal knowledge management case study. Journal of Knowledge Management, 2005.
16. Klein, M., Change Management for Distributed Ontologies. PhD thesis, Vrije Universiteit Amsterdam, 2004.