Computing Relaxed Answers on RDF Databases

Hai Huang[1], Chengfei Liu[1], and Xiaofang Zhou[2]

[1] Faulty of ICT, Swinburne University of Technology, Australia
{hhuang,cliu}@groupwise.swin.edu.au
[2] School of ITEE, The University of Queensland, Australia
zxf@uq.edu.au

Abstract. Database users may be frustrated by no answers returned when they pose a query on the database. In this paper, we study the problem of relaxing queries on RDF databases in order to acquire approximate answers. We address two problems for efficient query relaxation. First, to ensure the quality of answers, we compute the similarities of relaxed queries with regard to the original query and use them to score the potential relevant answers. We also propose the algorithm to get most relevant answers as soon as possible. Second, to optimise query relaxation process, we characterize a type of unnecessary relaxed queries which do not contribute to the final results and propose the method to prune them from the query relaxation graph. At last, we implement and experimentally evaluate our approach.

Keywords: RDF Database, RDF Query, Query Relaxation.

1 Introduction

Several RDF repositories have been developed in recent years such as Jena [1], Sesame [2], rdfDB [3]. Some of them can scale to million triples. With RDF databases growing in size and complexity, it becomes impractical to expect the users know enough about the contents and the structure of a database. So even when a user has a clear idea about the query condition, the database may still return an empty answer. In this case, the user has to change the query condition and try it again, which might become a "trial-and-error" process. Obviously, database users may be frustrated by this tedious process. The database system copes with it by relaxing the query condition automatically when the query fails to produce answers or enough answers the user expects. With query relaxation, the user can get exactly matched and non-exactly matched relevant answers.

A way of retrieving approximate answers is making the query condition more general in order to match potential answers. In RDF databases, structures do not come in the shape of well-defined schemas but in terms of semantic annotations that confirm to a schema called RDF schema (or RDFS ontology), which describes taxonomies of classes and properties. To enable relaxation, a query can be generalized to capture enough relevant answers using information provided by RDF schema.

For example, consider that a user wants to retrieve 10 proceedings editors who acquired doctoral degree from for University of Queensland (UQ) and work for

J. Bailey et al. (Eds.): WISE 2008, LNCS 5175, pp. 163–175, 2008.

Swinburne University (Swin) from the given RDF database. The user issues the following SPARQL query [4] on a database:

```
PREFIX   rdf:< http://www.w3.org/1999/02/22-rdf-syntax-ns# >
PREFIX   abc: < http://www.w3.org/abc.owl# >
SELECT ?X
WHERE {
   ?X   abc:doctoralDegreeFrom   abc:UQ.
   ?X   abc:worksFor     abc:Swin.
   ?X   abc:ProceedingsEditorOf   ?Y.
   ?Y   rdf:type   abc:Proceedings.
}
```

Fig. 1. A SPARQL Query

The system should return the exactly matched answers of the query. If no exactly matched answers return or the matched answers are not enough (less than 10), the system might relax the query condition using information in the RDFS ontology (defined in Fig.1) such as rewriting the condition (?X, doctoralDegreeFrom, UQ) to (?X, DegreeFrom, UQ) or (?Y, type, Proceedings) to (?Y, type, Book) for retrieving some potential relevant answers. Since the property "DegreeFrom" is more general than "doctoralDegreeFrom" and "Book" is the super class of "Proceedings" defined in the RDFS ontology.

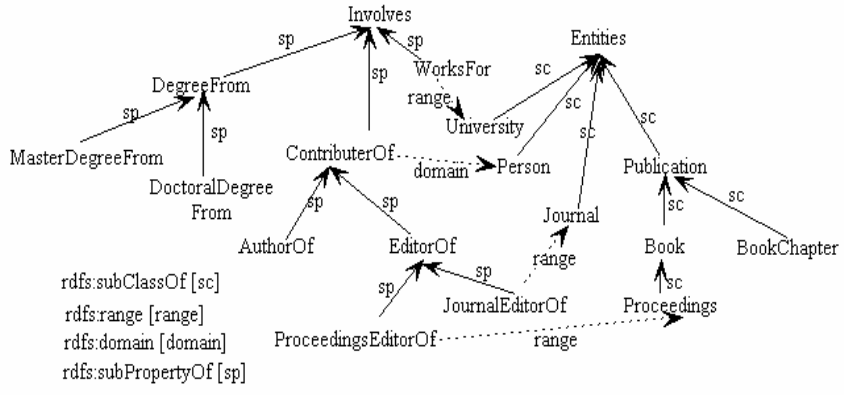

Fig. 2. A Sample of RDFS Ontology

From this example, two observations are in order. First, to guarantee the quality of answers, it is desirable to score the possible relaxed queries. Second, since several ways for relaxing the query condition may be available and the number of relaxations is huge, it is necessary to design an efficient relaxation algorithm that controls the query relaxation process and get most relevant answers as soon as possible. In this paper, we address these problems and make the following contributions:

(1) We measure the similarity degrees of the relaxed query with regard to the user query and use them to score the relevant answers.

(2) We design the algorithm to execute the relaxed queries according to their similarity score and return their answers incrementally.
(3) We characterize a type of *unnecessary relaxed queries* which do not contribute to the final results and propose the method to prune them from the query relaxation graph.

The remainder of this paper is organized as follows. We give the overview in section 2. We define the similarities of relaxed queries in section 3. Section 4 describes the relaxation algorithm and characterizes a type of unnecessary relaxed queries. Section 5 discusses related work. Section 6 presents an experimental evaluation of our approach.

2 Preliminary

2.1 Data and Query Model

A triple $t(s,p,o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple, where I is a set of IRIs (Internationalized URIs), B a set of blank nodes and L a set of literals. In a triple, s is called subject, p the property (or predicate), and o the object. RDF triples can be classified as schema triples (which describe the schema information) and data triples (which describe the instance data). In this paper, we consider data model where information is represented as a collection of RDF data triples stored in a relational database.

Each triple can also be represented as a node-arc-node link. The directed arc is the predicate and the nodes are its subject and object. A set of such triples is called an RDF graph. An RDF query is referred to as an RDF graph pattern, which consists of triple patterns. An **RDF triple pattern** $t(s,p,o) \in (I \cup V) \times (I \cup V) \times (I \cup V \cup L)$, where V is a set of variables disjoint from the sets I, B and L. **RDF graph pattern** $G = (q_1, q_2, ..., q_n)$, $q_i \in T$, where T is a set of triple patterns. Now we define the query model as follows.

Given a user query Q = (Atom (Q), K), where K is the number of answers that users desire and Atom (Q) = {<q_i, w_i>} defines the triple pattern with its weight w_i ($0<w_i<1$, default value of w_i is 1/m, where m is the number of triple patterns). For example:

$Q = \{< q_1, 0.2>, < q_2, 0.3>, < q_3, 0.3>, < q_4, 0.2>\}$
$q_1 = (?X, abc:phdDegreeFrom, abc:UQ)$
$q_2 = (?X, abc:worksFor, abc: Swin)$
$q_3 = (?X, abc:ProceedingsEditorOf, ?Y)$
$q_4 = (?Y, rdf:type Proceedings)$
$K=10$

2.2 Query Relaxation Model

Our techniques are based on logical relaxation of query conditions. In [5], the authors propose two kinds of relaxation for triple pattern which exploit RDF entailment to relax the queries.

Simple relaxation on triple pattern: Given RDF graphs G_1, G_2, a map from G_1 to G_2 is a function u from the terms (IRIs ,blank nodes and literals) in G_1 to the terms in G_2, preserving IRIs and literals, such that for each triple $(s_1, p_1, o_1) \in G_1$, we have $(u(s_1),$

$u(p_1)$, $u(o_1)) \in G_2$. This simple relaxation exploits the RDF simple entailment. AN RDF graph G_1 simply entails G_2, denoted by $G_1 \underset{simple}{\Rightarrow} G_2$, if and only if there is a map u from G_2 to G_1: $G_2 \xrightarrow{u} G_1 : \frac{G_1}{G_2}$. We call triple pattern t_2 the *simple relaxation* of t_1, denoted by $t_1 \underset{simple}{\prec} t_2$, if $t_1 \underset{simple}{\Rightarrow} t_2$ via a map u that preserves variables in t_1. For example, there is a map u from the terms of triple pattern (?X, type, ?Y) to (?X, type, Proceedings) that makes u("?X")= "?X", u("type")= "type" and u("?Y")= "Proceedings". So (?X, type, Proceedings) can be relaxed to (?X, type, ?Y) by replacing "Proceedings" with "?Y" and (?X, type, Proceedings) $\underset{simple}{\prec}$ (?X, type, ?Y).

Ontology relaxation on triple pattern: This type of relaxation exploits RDFS entailment in the context of an ontology (denoted by *onto*)**.** We call $G_1 \underset{RDFs}{\Rightarrow} G_2$, if G_2 can be derived from G_1 by iteratively applying rules in groups (A), (B) (sc, sp are rdfs:subclassOf and rdfs:subpropertyOf for short):

Group A (Subproperty) (1) $\frac{(a, sp, b)(b, sp, c)}{(a, sp, c)}$ (2) $\frac{(a, sp, b)(x, a, y)}{(x, b, y)}$

Group B (Subclass) (3) $\frac{(a, sc, b)(b, sc, c)}{(a, sc, c)}$ (4) $\frac{(a, sc, b)(x, type, a)}{(x, type, b)}$

Let t_1, t_2 be triple patterns, where $t_1 \notin$ closure (*onto*), $t_2 \notin$ closure (*onto*). We call t_2 an *ontology relaxation* of t_1, denoted by $t_1 \underset{onto}{\prec} t_2$, if $t_1 \cup onto \underset{RDFs}{\Rightarrow} t_2$. It includes relaxing type conditions and properties such as: (1) replacing a triple pattern (a, type, b) with (a, type, c), where (b, sc, c)$\in$ closure(*onto*). For example, given the ontology in Fig.2, the triple pattern (?X, type, Proceedings) can be relaxed to (?X, type, Book). (2) replacing a triple pattern (a, p_1, b) with (a, p, b), where (p_1, sp, p)$\in$ closure(*onto*). For example, the triple pattern (?X, ProceedingsEditorOf, ?Y) can be relaxed to (?X, EditorOf, ?Y).

Definition 1 ***(Relaxed Triple Pattern)*:** Given a triple pattern t, t' is a relaxed pattern obtained from t, denoted by $t \prec t'$, through applying a sequence of zero or more of the two types of relaxations: simple relaxation and ontology relaxation.

For example, the triple pattern (?X,ProceedingsEditorOf, ?Y) $\prec$ (?X, Contributer Of , ?Y)

Definition 2 ***(Relaxed Query Pattern)*:** Given a query pattern Q (t_1, t_2,...,t_n), Q' (t_1', t_2'... t_n') is the relaxation of the Q, denoted by $Q \prec Q'$, if there exists at least one pair (t_i, t_i') that we have $t_i \prec t_i'$.

Definition 3 ***(Relevant Answers)*:** A relevant answer to the query Q is defined as a match of a relaxed query of Q.

A user query can be relaxed through replacing a term in the query by a more general term or replacing values to variables. For instance, a query Q (shown in Fig.1) has relaxed queries such as Q'={(?X, DegreeFrom, UQ),(?X, worksFor, Swin),

(?X, ProceedingsEditorOf, ?Y),(?Y, type, Proceedings)}. Q″={(?X, dotoralDegree-From, UQ),(?X, worksFor, Swin),(?X, ProceedingsEditorOf, ?Y),(?Y, type, Book)}. Obviously, there are many different ways to relax a query and the amount of relaxations will be large. Returning all answers of these relaxed queries is not desirable. So we rank the relaxed queries based on their similarities to the original query and execute the most similar relaxed queries first in order to achieve most relevant answers. Given the user query Q and the relaxed queries Q′, Q″ of query Q, If Q′ is more similar to Q than Q″, then we return the results Q′ prior to the results of Q″. Thus, the query relaxations require measuring the similarities of the relaxed queries with regard to the original query.

3 Similarities of Relaxed Queries

Since the amount of possible query relaxations may be large, blindly relaxing query would not be helpful for getting the most relevant answers. To guarantee the quality of answers, it is desirable to measure the similarities of the relaxed query to the user query and choose the most similar one to execute on the database first. An RDF query is referred to as a graph pattern, which consists of triple patterns. Thus we compute the similarities of the relaxed triple patterns and then combine them to compute the similarities of the relaxed queries.

3.1 Similarities of Triple Patterns

Note that an RDF triple pattern can be represented as a node-arc-node link with two nodes (subject, object) and one arc (property). Hence we introduce the similarity between nodes as well as arcs and integrate them to define our similarity of the relaxed triple patterns. We define a finite set of class names *IC* and property names *IP*.

Similarity between nodes: In the triple pattern q_1, if one node (subject or object) is a class $c_1 \in IC$, defined in the RDFS ontology, and it is relaxed to its super class c_2 through ontology relaxation we exploit the notion of the Least Common Ancestor (LCA) to compute the similarity of two nodes based on the ontology. The LCA of two nodes c_1 and c_2 in the IS-A hierarchy is the node that is an ancestor of both c_1 and c_2, where depth of a concept node is the maximum length of from the concept to the root of the taxonomy.

$$sim(c_1, c_2) = \frac{2 \times depth(LCA(c_1, c_2))}{depth(c_1) + depth(c_2)}$$

For example, Sim("Book","Proceedings")=2×2/(2+3)=0.8. If the node (subject or object) in the triple pattern is relaxed to a variable through simple relaxation, we define the similarity of the two nodes is *0*. For example, (?X, type, Proceedings) is relaxed by (?X, type, ?W), we have *sim*("Proceedings","?W")=0.

Similarity between arcs: If the property $P_1 \in IP$ in the triple pattern is relaxed to its super property P_2 through ontology relaxation, since the hierarchy (sub property relationship) on properties is also defined in the RDFS ontology, the similarity definition on nodes can be transferable to arcs. We have:

$$sim(p_1, p_2) = \frac{2 \times depth(LCA(p_1, p_2))}{depth(p_1) + depth(p_2)}$$

$LCA(p_1,p_2)$ indicates the least common super property of p_1 and p_2. And the depth of a property node is the maximum length from the property to the root of the sub property taxonomy.

For example, *sim*("proceedingsEditorOf","EditorOf")=2×2/(2+3)=0.8 If the predicate in the triple pattern is relaxed to a variable through simple relaxation, we define the similarity of the two arcs as 0. For example, (?X, doctoralDegreeFrom, UQ) is relaxed to (?X, ?W, UQ), then we have sim("doctoralDegreeFrom" , "?W")=0.

Similarity of triple patterns: Given a triple q(s, p, o) and its relaxation q′(s′, p′, o′), we integrate similarity between nodes and arcs to define the similarity score of the relaxed triple patterns as follows:

$$score(q, q') = sim(s, s') + sim(p, p') + sim(o, o') \quad (1)$$

3.2 Similarities of Relaxed Queries and Relevant Answers

Given a query pattern Q (q_1, q_2,…,q_n) and its relaxation Q′ (q_1', q_2',…,q_n'), we can compute the semantic similarity *comScore* between them as follows:

$$comScore(Q, Q') = \sum_{i=1}^{n} w_i \times score(q_i, q_i') \quad (2)$$

Where w_i is the weight of triple pattern q_i in the query pattern Q. The scoring function *comScore* is *monotone* in the sense that if Q′ (q_1', q_2',…,q_n') and Q″ (q_1'', q_2'',…,q_n'') are the relaxations of query Q and $score(q_i, q_i') \geq score(q_i, q_i'')$, then $comScore(Q, Q') \geq comScore(Q, Q'')$.

In general, we can define the score of a relevant answer as the similarity of relaxed the query it matches. However, a relevant answer may match the different relaxed queries, e.g., ("John", AuthorOf, "WISE conference") would match queries (?X, AuthorOf, "WISE conference") and (?X, ContributerOf, "WISE conference"). To deal with this situation, we define the score of a relevant answer as follows:

Definition 4 (Score of a relevant answer): The score of a relevant answer is the maximum *comScore* among all of the relaxed queries it matches.

4 Query Relaxation Processing

In this section, we focus on properly relaxing a user query that produces a sequence of relaxed queries based on their similarity scores. To achieve it, the query relaxation algorithm is presented. We also characterize a type of "unnecessary relaxed queries" which do not contribute to the final results and propose the method to prune them. At last, the execution algorithm on a database is proposed.

4.1 Relaxation Algorithm

We design the relaxation algorithm that ensures the desired quality of answers. The relaxed queries are executed in a sequence based on their similarities with regard to the original query. For instance, Fig.1 shows the query Q={(?X, doctoralDegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Proceedings)} and it has several relaxed queries, such as: Q1={(?X, DegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Proceedings)}. Q_4={(?X, dotoralDegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Book)}. Since $comScore(Q,Q_1)=0.964>comScore(Q,Q_4)=0.944$ (shown in Fig.3), which means that query Q_1 is more similar to the user query and hence should be executed first. We now present a query relaxation strategy, which selects the next most similar relaxed query to execute on the database till enough results are returned. We first construct the relaxation graph and then give the relaxation algorithm based on this graph.

Ordering the relaxations of triple patterns: Given the user query denoted by $Q(q_1^{(0)},q_2^{(0)},\ldots,q_n^{(0)})$, we compute the relaxations of each triple pattern and rank them by their similarity scores. $q_i^{(0)}$ indicates the original triple pattern and $q_i^{(m_i)}$ indicates the m_ith relaxation of triple pattern $q_i^{(0)}$. An example is shown in Table 1. If the original triple pattern $q_i^{(0)}$ is relaxed to $q_i^{(m_i)}$, we say that the triple pattern $q_i^{(0)}$ goes forward m_i steps. Obviously, if $m_i \le m_j$, then $score(q_i^{(0)},q_i^{(m_i)}) \ge score(q_i^{(0)},q_i^{(m_j)})$.

Relaxation Graph: The relaxation graph is a directed graph representing the relaxed queries. Each node in the relaxation graph is a relaxed query. Q_2 is Q_1's *succeeding node* if the corresponding triple patterns in two queries are same except only one triple pattern in Q_1 that is relaxed one step to the corresponding triple pattern in Q_2. Q_1 is also called Q_2's *preceding node*. For instance, $Q_3(q_1^{(0)},q_2^{(0)},q_3^{(1)},q_4^{(0)})$ is $Q(q_1^{(0)}$

Table 1. Ranking of the relaxed triple patterns of query Q with scores

$q_1^{(0)}$:(?X,doctoralDegreeFrom, UQ) ; 1 $q_1^{(1)}$:(?X, degreeFrom, UQ); 0.82 $q_1^{(2)}$:(?X,doctoralDegreeFrom?W_1); 0.54 …	$q_2^{(0)}$:(?X,worksFor,Swin); 1 $q_2^{(1)}$:(?X,worksFor,?W_3); 0.55 $q_2^{(2)}$:(?X,?W_4,Swin); 0.5 …	$q_3^{(0)}$:(?X,ProceedingsEditorOf, ?Y); 1 $q_3^{(1)}$:(?X,EditorOf,?Y); 0.7 $q_3^{(2)}$:(?X,CotributerOf,?Y); 0.6 …	$q_4^{(0)}$(?Y,type,Proceedings); 1 $q_4^{(1)}$(?Y,type,Book);0.72 $q_4^{(2)}$(?Y,type,Publication); 0.65 …

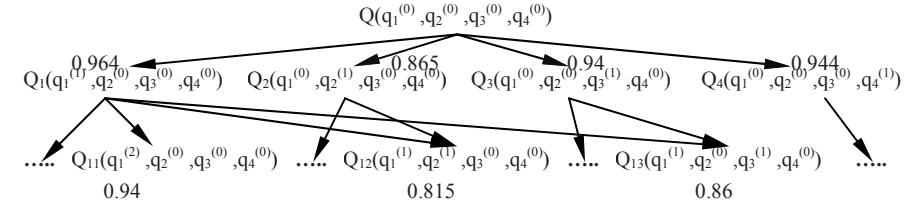

Fig. 3. Relaxation graph of query Q

$q_2^{(0)}$,$q_3^{(0)}$,$q_4^{(0)}$)'s succeeding node. An *edge* from node Q_1 to Q_2 exists if and only if Q_2 is Q_1's succeeding node. The relaxed queries can be categorized into different levels. For a relaxed query $Q_j(q_1^{(m_1)}, q_2^{(m_2)}, \ldots, q_n^{(m_n)})$, if $m_1+m_2+\ldots+m_n=h$, then Q_j is in the level *h*. An example of query relaxation graph is shown in Fig.3. Note that the relaxation graph we consider has several interesting properties, which will serve as the bases for query relaxation algorithm.

Property 1. Given a relaxed query $Q_j(q_1^{(m_1)}, q_2^{(m_2)}, \ldots, q_n^{(m_n)})$ and its succeeding nodes $Q_{j1}(q_1^{(m_1+1)}, q_2^{(m_2)}, \ldots, q_n^{(m_n)})$, $Q_{j2}(q_1^{(m_1)}, q_2^{(m_2+1)}, \ldots, q_n^{(m_n)}), \ldots, Q_{jn}(q_1^{(m_1)}, q_2^{(m_2)}, \ldots, q_n^{(m_n+1)})$, we have $comScore(Q,Q_j) \geq comScore(Q,Q_{ji})$, $i \in [1, n]$, and at least one relaxed query in level *h* has higher *comScore* than all relaxed queries in level $h' > h$.

Proof. Let query $Q_j(q_1^{(m_1)}, \ldots, q_i^{(m_i)}, \ldots, q_n^{(m_n)})$ and $Q_{ji}(q_1^{(m_1)}, \ldots, q_i^{(m_i+1)}, \ldots, q_n^{(m_n)})$ be the relaxations of query $Q(q_1^{(0)}, q_2^{(0)}, \ldots, q_n^{(0)})$ where Q_{ji} is a succeeding relaxation node of Q_j, it holds that $score(q_i^{(0)}, q_i^{(m_i)}) \geq score(q_i^{(0)}, q_i^{(m_i+1)})$. By monotonicity of scoring function *comScore*, we have $comScore(Q,Q_j) \geq comScore(Q,Q_{ji})$. It shows that the *comScore* of one relaxed query is higher than its succeeding nodes. So the relaxation node with maximum *comScore* in level *h* must has higher *comScore* than all relaxed queries in level $h' > h$. □

Property 2. Given the original query $Q(q_1^{(0)}, q_2^{(0)}, \ldots, q_n^{(0)})$, the relaxed query Q′ with the *h*th highest *comScore* must be in the level *h* or less.

Proof. When x=1, the relaxed query with the highest comScore must fall in level 1 since property 1. We assume that when x=h, it holds that the relaxed query with the *h*th highest comScore falls in level h. When x =h+1, we can find at least one relaxed query in level h+1 has higher comScore than all relaxations in level h′>h+1 according to property 1. So when x =h+1, we can get h+1 relaxed queries within h+1 level. □

We exploit Property 1 and Property 2 to define the following basic relaxation algorithm that generates the next most similar relaxed queries incrementally. From property 2, the relaxed query with the highest *comScore* must be in the first level. Thus we first get the succeeding nodes of the original query and choose the query with the highest *comScore* to execute on the database. If the number of answers is not enough, then we choose the next relaxed query with the second highest *comScore* from the first or second level.

Basic Relaxation Algorithm /* Input: $Q(q_1^{(0)}, q_2^{(0)}, \ldots, q_n^{(0)})$, *K* the number of results required ; Output : *Result* */

```
1   Initial Result= Φ, Candidates= Φ, Processed= Φ;
2   Insert Q's succeeding nodes into Candidates;
3   While | Result |<K and |Candidates| >0 do
4       Select the Qi with the highest comScore from Candiates
5       Insert Qi's succeeding nodes into Candidates;
6       R←Execute (Qi);
7       Result ←Result ∪ R;
8       Add Qi to Processed;
9       Remove Qi from Candiates ;
10  End while
11  Return Result;
```

For example, the user query Q{(?X, doctoralDegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Proceedings)}(in Fig.1) is executed and the query Q's succeeding relaxed queries Q_1-Q_4 are added into *Candidates*. If the number of answers is still not enough, we choose the relaxed query Q_1{(?X, DegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Proceedings)} that has the highest *comScore* in *Candidates* and execute it on the database. We also add Q_1's succeeding relaxed queries into *Candidates*, and remove Q_1 from *Candidates*. This process is repeated till enough answers are acquired or no more candidate is left.

4.2 Pruning Unnecessary Relaxations

In the basic relaxation algorithm above, when a relaxed query is selected, we add its succeeding nodes (relaxations) into *Candidates* for comparison of next round. However, the succeeding relaxations may be useless to generate new answers compared to the preceding relaxed query and there are unnecessary relaxations. For instance, in our running example, the query Q has one of its relaxation Q_4{(?X, DegreeFrom, UQ), (?X, worksFor, Swin), (?X, ProceedingsEditorOf, ?Y), (?Y, type, Book)}, which replaces the term "Proceedings" in query Q to "Book" only. However, this relaxation Q_4 will not generate new answers compared to the answers of its preceding node (query Q). Because the triple (?Y, type, Proceedings) has join dependency with the triple pattern (?X, ProceedingsEditorOf, ?Y). In the relaxed query Q_4 "Proceedings" is relaxed to "Book", but the domain of property "ProceedingsEditorOf "is still "Proceedings" (according to the RDFS ontology defined in Fig.2) and this relaxation will generate no new answers compared to the answers previously returned by the query Q. The basic algorithm relaxes the classes or properties in the triple patterns of the query *locally*. It fails to consider the join dependency between the triple patterns through common variables. Consequently, some relaxed queries may not return new answers. There are two cases of unnecessary relaxations that fail to contribute new answers, one in generalizing the subject or object term of a triple pattern, while the other in generalizing the property.

We first define the boolean operator "$\leq$"on classes and properties. If class c_1 is the subclass of c_2, we have $c_1 \leq c_2$. Similarly, if property p_1 is the sub property of p_2, we have $p_1 \leq p_2$.We also define the operator "*subclass*(*C*)" which computes the set of sub classes of *C* and "subproperty(P)", which computes the set of sub properties of *P*.

Case 1: Given a query Q $\{q_1,\ldots,q_i(?x, type, c_1),\ldots,q_j(?x, p_2, o_2)$ or $(s_2, p_2, ?x),\ldots,q_n\}$, its succeeding relaxation Q′ $\{q_1,\ldots,\underline{q_i'(?x, type, c)},\ldots,q_j(?x, p_2, o_2)$ or $(s_2, p_2, ?x),\ldots,q_n\}$ is obtained from query Q through replacing class c_1 to c, where $c_1 \leq c$, p_2 is a property and $o_2 \in (I \cup V \cup L)$, $s_2 \in (I \cup V)$.

$$\Delta\omega = subclass(c) - subclass(c_1)$$

If $\exists\xi \in \Delta\omega$ and $\xi \leq domain(p_2)$(or $range(p_2)$), then the relaxed query may have new answers generated. Otherwise, the relaxed query has no new answer generated compared to query Q.

When generalizing the property of a triple pattern in the query, new answers could be obtained through matching the generalized property or its sub properties. Because of join dependency between triple patterns, the domain (subject) and range (object) of the generalized property or its sub properties may not match with other properties in the query. This is formalized as follows:

Case 2: Given a query $Q\{q_1,\ldots,q_i(?x, p_1, o_1) ,\ldots,q_j(?x, p_c, o_2),\ldots,q_n\}$, its succeeding relaxation $Q'\{q_1,\ldots,\underline{q_i'(?x, p, o_1)},\ldots,q_j(?x, p_c, o_2),\ldots,q_n\}$ is obtained from the query Q through replacing the property p_1 with p, where $p_1 \leq p$; p_c is a property and $o_1, o_2 \in (I \cup V \cup L)$. After relaxation, new answers could be added through matching the property p or its sub properties. Notice that $q_i'(?x, p, o_1)$ has join dependency with $q_j(?x, p_c, o_2)$ through variable "?x". If $D=subproperty(p)\text{-}subproperty(p_1)$ and $(\bigcup_{p_i \in D} subclass(domain(p_i))) \cap (subclass(domian(p_c))) = \Phi$, then the relaxed query Q′ has no new answer generated. It is straightforward to generalize the method to the situation that q_i and q_j take the form $(s_1, p_1, ?x)$ and $(s_2, p_c, ?x)$.

Now we give the optimised relaxation algorithm, which checks the usefulness of new relaxed query and skip those unnecessary relaxed queries.

Optimised Relaxation Algorithm /* Input: $Q(q_1^{(0)}, q_2^{(0)}, \ldots, q_n^{(0)})$, *K* the number of results required ; Output : *Result* */

```
1   Initial Result= Φ, Candidates= Φ, Processed= Φ;
2   Insert Q's succeeding nodes into Candidates;
3   While | Result |<N and Candidates| >0 do
4       Select the Qi with the highest comScore from the Candiates;
5       Insert Qi's succeeding nodes into Candidates;
6       For each of Qi's preceding nodes Qj in Processed
7           If   Qi is an unnecessary relaxation of Qj then
8               Goto step 13
9           End if
10      End for
11      R←Execute (Qi);
12      Result ←Result  ∪ R;
13      Add Qi to Processed;
14      Remove Qi from Candiates ;
15  End while
16  Return Result
```

5 Related Work

Hurtado et al. [5] propose an rdf query relaxation method through RDF(s) entailment producing more general queries for retrieving potential relevant answers. Our work focuses on using heuristic information based on the similarity degree of relaxed queries with regard to the original query to control the relaxation process for ensuring the desired cardinality and quality of answers, which is not studied in [5].

Answering top-k selection queries is related to our work. Fagin et al. [6, 7] introduce a set of novel algorithms, assuming sorted access and/or random access of the objects is

available on each attribute. Carey and Kossmann [8] optimise top-k queries when the scoring is done through a traditional SQL order by clause, by limiting the cardinality of intermediate results. Other works [9, 10, and 11] use statistical information to map top-k queries into selection predicates which may require restarting query evaluation when the number of answers is less than K. Natsev et al. [12] introduce the J* algorithm to join multiple ranked inputs to produce a global rank. These works define the semantics of the relevant answers based on numeric conditions in selection and join predicates, which can be quantified as value differences. In contrast, we relax query conditions and rank answers based on semantic information provided by RDF ontology.

Cooperative query answering [13, 14, 15] are designed to automatically relax user queries when the selection criteria is too restrictive to retrieve enough answers. Such relaxation is usually based on user preferences and values. In contrast, in our work, we compute the semantic similarities between the relaxed queries and the original query. Efforts have been made to the problem of query relaxation on XML data [16]. Amer-Yahia et al. [16] compute approximate answers for weighted patterns by encoding the relaxations in join evaluation plans. The techniques of approximate XML query matching are mainly based on structure relaxation and can not be used to handle query relaxation on RDF data directly.

6 Experiments

Based on the Lehigh University Benchmark LBUM [17], we generate the dataset which contains 6,000k triples. The data is stored in and managed by Mysql 5.0.11. All algorithms are implemented using Jena SDB [18] (which provides for large scale storage and query of RDF datasets using conventional SQL databases) and run on a windows XP professional system with P4 3G CPU and with 512 M RAM. We developed 5 queries which are shown in Table 2.

Benefits of the optimised algorithm: In this experiment, we compare the basic relaxation algorithm and optimised algorithm. We fix the number of answers K=100 and performed the 5 queries on the data which contains 6,000k triples. Fig.4 (a) and (b) show the relaxation steps and execution time for each query. We use query Q_4 as an example to illustrate how the optimisation algorithm reduces the relaxation steps.

Table 2. Queries for experiments

Q_1:(?x,type,TeachingAssitant)(?x,teachingAssistantOf,http://www.Department0.University0/Course3)(?x,mastersDegreeFrom,http://www.Department0.University0.edu)
Q_2:(?x,teacherOf,?z)(?x,ub:worksFor,University0)(?x,type,AssistantProfessor)
Q_3:(?x,advisor,?y)(?y,type,AssistantProfessor)(?y,researchInterest',Research12')(?y,worksFor,http://www.University0.edu)
Q_4:(?x,advisor,?y)(?y,type,Professor)(?y,worksFor, http://www.University0.edu)
Q_5:(?x,type,JournalAticle)(?y,publicationAuthor,?x) (?y,type,Professor)

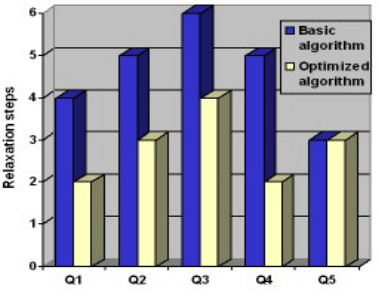

Fig. 4. (a). Relaxation Steps

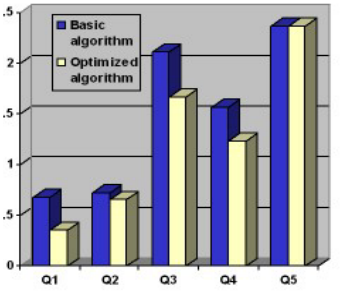

Fig. 4. (b). Query Response Time

Query Q_4 is relaxed to its relaxation Q_4' through replacing *"Professor"* to *"Person"* only. In Q_4' the range of property *"advisor"* is still *"Professor"*. So only replacing *"Professor"* to *"Person"* will not generate new answers compared to the answers of query Q_4 and Q_4' is a unnecessary relaxation to query Q_4. Fig.4 (a) and (b) show that the optimisation algorithm can reduce the relaxation steps and execution time.

Efficiency of query relaxation: Fig.4 (b) shows that the running time of 5 queries for K=100 relaxed answers. Q_5 is the most expensive query. Because the query condition of Q_5 is more general and with the increase of relaxation steps the relaxation process costs much. Q_1 is the most efficient one since its query condition is more concrete. It takes less running time than Q_5, though the relaxation steps of Q_1 are less than that of Q_5. Moreover, our relaxation algorithm can return relevant answers incrementally and users can stop the relaxation process at any time when they are satisfied with the answers generated.

7 Conclusion

This paper addressed the issue of relaxing the user query on RDF databases and computing most relevant answers. We measured the similarity degrees of the relaxed queries with regard to the user query and designed the algorithm to retrieve the most relevant answers as soon as possible. We characterized a type of unnecessary relaxed queries which do not contribute to the final results and proposed the method to prune them from the query relaxation graph. The experiments validated our approach. A further optimisation is subject to our future work such as multiple query optimisation on a sequence of relaxed queries by reusing the intermediate results of selection or join operations.

References

1. Jena, `http://jena.sourceforge.net/`
2. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: International Semantic Web Conference, pp. 54–68 (2002)

3. Guha, R.: rdfDB: An RDF Database, http://www.guha.com/rdfdb/
4. SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
5. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: A Relaxed Approach to RDF Querying. In: International Semantic Web Conference, pp. 314–328 (2006)
6. Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In: PODS (2001)
7. Fagin, R.: Fuzzy Queries in Multimedia Database Systems. In: PODS, pp. 1–10 (1998)
8. Carey, M.J., Kossmann, D.: On Saying "Enough Already!" in SQL. In: SIGMOD Conference, pp. 219–230 (1997)
9. Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. ACM Trans. Database Syst. 27(2), 153–187 (2002)
10. Chen, C.-M., Ling, Y.: A Sampling-Based Estimator for Top-k Query. In: ICDE, pp. 617–627 (2002)
11. Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In: SIGMOD Conference, pp. 259–270 (2001)
12. Natsev, A., Chang, Y.-C., Smith, J.R., Li, C.-S., Vitter, J.S.: Supporting Incremental Join Queries on Ranked Inputs. In: Proc. of the 27th International Conference on Very Large Data Bases, pp. 281–290 (2001)
13. Godfrey, P.: Minimization in Cooperative Response to Failing Database Queries. Int. J. Cooperative Inf. Syst. 6(2), 95–149 (1997)
14. Chu, W.W., Yang, H., Chiang, K., Minock, M., Chow, G., Larson, C.: CoBase.: A Scalable and Extensible Cooperative Information System. J. Intell. Inf. Syst. 6(2/3), 223–259 (1996)
15. Kleinberg, J.M.: Authoritative Sources in a Hyperlinked Environment. J. ACM 46(5), 604–632 (1999)
16. Amer-Yahia, S., Cho, S., Srivastava, D.: Tree Pattern Relaxation. In: Jensen, C.S., Jeffery, K.G., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 496–513. Springer, Heidelberg (2002)
17. Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In: International Semantic Web Conference, pp. 274–288 (2004)
18. Jena SDB, http://jena.hpl.hp.com/wiki/SDB