Query Relaxation for Star Queries on RDF

Hai Huang and Chengfei Liu

Faculty of ICT, Swinburne University of Technology
VIC 3122, Australia
{hhuang,cliu}@swin.edu.au

Abstract. Query relaxation is an important problem for querying RDF data flexibly. The previous work mainly uses ontology information for relaxing user queries. The ranking models proposed, however, are either non-quantifiable or imprecise. Furthermore, the recommended relaxed queries may return no results. In this paper, we aim to solve these problems by proposing a new ranking model. The model ranks the relaxed queries according to their similarities to the original user query. The similarity of a relaxed query to the original query is measured based on the difference of their estimated results. To compute similarity values for star queries efficiently and precisely, Bayesian networks are employed to estimate the result numbers of relaxed queries. An algorithm is also proposed for answering top-k queries. At last experiments validate the effectiveness of our method.

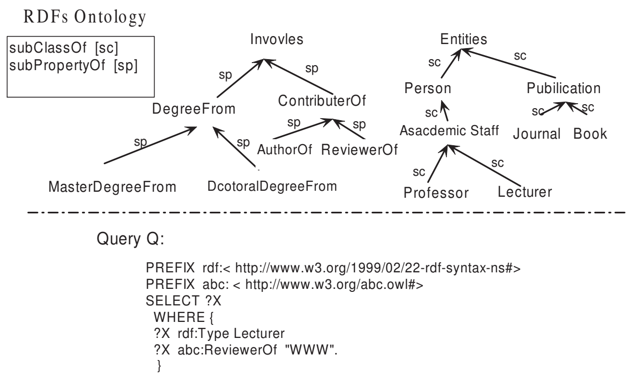
Key words: Query Relaxation, RDF query Processing

1 Introduction

The Resource description Framework (RDF) is a data model used on the Semantic Web. Recently, more and more data is represented and stored in RDF format. RDF data is a set of triples and each triple called statement is of the form (*subject*, *property*, *object*). This data representation is general and flexible. Almost any kind of data can be represented in this format. To query RDF data, many query languages on RDF data have been proposed and implemented such as SPARQL. SPARQL is an RDF query language, which has a SQL-like style. A SPARQL query usually consists of several triple patterns.

With RDF data growing in size and complexity, several RDF repositories have also been developed in recent years such as Jena, Sesame. Generally, users do not know the whole RDF database and often post the *failure queries*, which do not return any answers. In this case, it is desirable to relax the original user query to obtain some approximate answers. Given the RDFs ontology, a failure query can be relaxed to more general queries.

For example, Fig.1 shows an RDFs ontology and a SPARQL query Q, which retrieves the academic staff members who are lecturers and also the reviewers of conference "WWW". Suppose Q returns no answers (or not enough answers) and we try to relax this query to achieve some approximate answers. Using the ontology information in Fig.1, query Q can be generalized to capture approximate



```
PREFIX  rdf:< http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  abc: < http://www.w3.org/abc.owl#>
SELECT ?X
  WHERE {
  ?X  rdf:Type Lecturer
  ?X  abc:ReviewerOf  "WWW".
  }
```

Fig. 1. The Example of RDFs Ontology and a SPARQL Query Q

answers in many ways. Queries Q'(?X, type, AcademicStaff)(?X, ReviewerOf, "WWW") and Q''(?X, Type, Lecturer)(?X, ReviewerOf, ?Y) are both the relaxed queries of Q. The answers of each relaxed query are approximate answers of Q. Since the number of possible relaxations is huge, we hope to rank them.

To rank these relaxations, the authors in [1] introduced the ranking model on the relaxed queries that have the *subsume* relationship among them. Given two relaxations Q_i and Q_j of the original query Q, if Q_i is subsumed by Q_j, then Q_i is better. However, this ranking model is non-quantifiable; moreover, it fails to rank Q' and Q'' in the above example because they have no subsume relationship between them.

In [5], Huang and Liu proposed a method to rank the relaxed queries depending on the extent to which the concepts and properties in the relaxed queries are similar to the user query. First, this ranking model is a little coarse-grained because it is only based on the ontology information. Furthermore, the ranking model does not consider the real data distribution in the database, so it is possible that relaxed queries may still have no answers in the database, which would lead to unnecessary execution cost. To solve this problem, we propose a fine-grained ranking model and consider the selectivity of relaxed queries in our ranking model.

In this paper, we propose new methods to rank the relaxed queries and compute the top-k approximate answers. The contributions of this paper can be summarized as follows:

- We construct a ranking model which ranks the relaxed queries according to their similarities to the original user query. The similarity of a relaxed query to the original query is measured based on the difference of their estimated results.

- We propose a method to compute the similarity values of relaxed star queries efficiently using Bayesian networks.

– We develop an algorithm to compute the top-k approximate answers according to our ranking model based on best-first search.

The remainder of this paper is organized as follows. Section 2 introduces some preliminary knowledge. In Section 3, we construct the ranking model. Section 4 presents the method to compute the similarity value of relaxed queries efficiently using Bayesian Networks. In Section 5 we present the algorithm to compute the top-k approximate answers for queries. Section 6 describes an experimental evaluation of our approach. Some related work is discussed in Section 7. At last, in Section 8, we conclude our work in this paper and discuss the future work.

2 Preliminary

2.1 Basic concepts and problem definition

A *triple* $(s, p, o) \in (I \cup B) \times (I \cup B) \times (I \cup B \cup L)$ 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 the triple, s is called subject, p the property (or predicate), and o the object or property value. An RDF *triple pattern* $(s, p, o) \in (I \cup V) \times (I \cup V) \times (I \cup V \cup L)$, where V is a set of variables disjoint from the sets I, B and L. An *RDF graph pattern* $G = (q_1, q_2, ..., q_n), q_i \in T$, where T is a set of triple patterns.

In this paper, we focus on star SPARQL query patterns, which are common in SPARQL query patterns. The star query pattern retrieves instances of a class with some conditions. Fig.1 shows an example of star SPARQL query pattern.

Definition 1 (Star Query patterns): The star query pattern has the form of a number of triple patterns with different properties sharing the same subject.

2.2 Query Relaxation Model

Hurtado *et al.* in [1] 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 $G_1 \underset{\text{simple}}{\Longrightarrow} G_2$, if and only if there is a map u from G_2 to G_1: $G_2 \overset{u}{\to} G_1 : \frac{G_1}{G_2}$. We call triple pattern t_2 the simple relaxation of t_1, denoted by $t_1 \underset{\text{simple}}{\prec} t_2$, if $t_1 \underset{\text{simple}}{\Longrightarrow} t_2$ 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, Lecturer) that makes u(”?X”)= ”?X”, u(”type”)= ”type” and u(”?Y”)= ”Lecturer”. So (?X, type, Lecturer) can be relaxed to (?X, type, ?Y) by replacing ”Lecturer” with the variable ”?Y”. We have (?X, type, Lecturer) $\underset{\text{simple}}{\prec}$ (?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 $G_1 \overset{\Longrightarrow}{\scriptstyle \text{RDFs}} G_2$, if G_2 can be derived from G_1 by iteratively applying rules in groups (A), (B) (C)(sc, sp are rdfs:subclassOf and rdfs:subpropertyOf for short):

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

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

– Group C (Typing) (5) $\frac{\text{(a dom c)(x a y)}}{\text{(x type c)}}$; (6) $\frac{\text{(a range d)(x a y)}}{\text{(y type d)}}$

Let t, t' be triple patterns, where $t \notin closure(onto)$, $t' \notin closure(onto)$. We call t' an ontology relaxation of t, denoted by $t \overset{\prec}{\scriptstyle \text{onto}} t'$, if $t \cup onto \overset{\Longrightarrow}{\scriptstyle \text{RDFs}} t'$. 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.1, the triple pattern (?X, type, Lecturer) can be relaxed to (?X, type, AcademicStaff). (2) replacing a triple pattern (a, p1, b) with (a, p, b), where (p1, sp, p) $\in$ *closure(onto)*. For example, the triple pattern (?X, ReviewerOf, "WWW") can be relaxed to (?X, ContributerOf, "WWW"). (3) replacing a triple pattern (x, p, y) with (x, type, c), where (p, domain, c) $\in$ *closure(onto)*. For example, the triple pattern (?X, ReviewerOf, "WWW") can be relaxed to (?X, Type, Person).

Definition 2 (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.*

Definition 3 (Relaxed Query Pattern): Given a user query $Q(q_1, q_2, \cdots, q_n)$, $Q'(q'_1, q'_2, \cdots, q'_n)$ is a relaxed query of Q, denoted by $Q \prec Q'$, if there exists $q_i = q'_i$ or $q_i \prec q'_i$ for each pair (q_i, q'_i).

Definition 4 (Approximate Answers): An approximate answer to query Q is defined as a match of a relaxed query of Q.

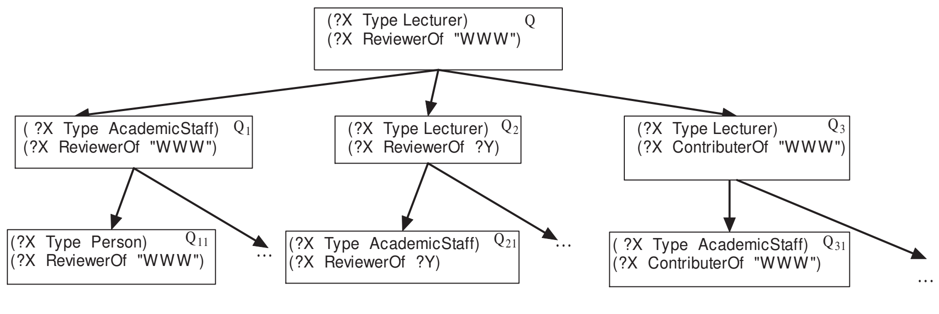


Fig. 2. Query Q and its relaxation graph.

3 Ranking Model for Relaxations

User queries can be relaxed in many ways by ontology relaxation and simple relaxation. For instance, in Fig.2, query Q has relaxed queries Q_1=(?X, Type, AcademicStaff)(?X, ReviewerOf, "WWW"), Q_2=(?X, Type, Lecturer)(?X, ReviewerOf, ?Y) and so on. Since the number of relaxations could be large, returning all answers of these relaxed queries is not desirable. So we need to rank the relaxed queries and execute the better one first. Given the user query Q and its relaxed queries Q_1, Q_2, if Q_1 is ranked prior to Q_2, then we return the results Q_1 prior to the results of Q_2. In this section, we discuss the ranking model for relaxed queries.

Intuitively, for a relaxation Q' of user query Q, the more similar Q' is to Q, the better Q' is. Thus, we hope to rank the relaxed queries based on their similarities to the original query. Several similarity measures for concepts have been proposed such as information content measure [4], distance based measure [6]. Each method assumes a particular domain. For example, distance based measure is used in a concept network and Dice coefficient measure is applicable when the concepts can be represented as numerical feature vectors. To our problem, the goal is to measure similarity between Q and its relaxations Q_i and our objects are queries not concepts. Thus, these similarity measures can not be directly applied in our problem. We need to find an appropriate method to measure the similarity between the relaxed queries and original query. Before giving the method to quantify the similarity value, we first clarify some helpful intuitions as follows:

Intuition 1: The similarity between Q and its relaxation Q_i is proportional to their commonality. From Fig. 3, we can see that the more answers Q and Q_i share in common, the more similar they are.

Intuition 2: The similarity between Q and its relaxation Q_i ranges from 0 to 1 and if Q and Q_i are identical (i.e., they have the same result set), the similarity value should be 1.

Intuition 3: Given Q and its relaxation Q_i and Q_j, if $Q_i \prec Q_j$, then Q_i should be more similar to Q. It indicates that the ranking model should be consistent.

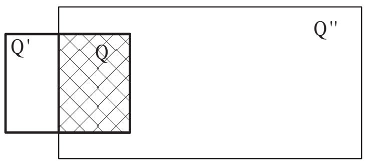


Fig. 3. Query Q and its relaxations Q' and Q''. The shaded area shows the answers they share in common.

We know that the information contained by a query Q is its answers in the database. We define $p(Q)$ to be the probability of encountering an answer of query Q in the database. According to information theory, the *information content* of query Q, $I(Q)$ can be quantified as the negative log likelihood, $I(Q) = -log\ p(Q)$. For query Q and its relaxation Q_i, the information shared by two queries is indicated by the information content of the query that subsume them. Here, since $Q \prec Q_i$, we have:

$$Common(Q, Q_i) = I(Q_i) = -log\,\Pr(Q_i)$$

Where $\Pr(Q)$ stands for the probability of encountering an answer of query Q in the database. And we have:

$$\Pr(Q) = \frac{|Ans(Q)|}{|Res|}$$

where $Ans(Q)$ denotes the answer set of query Q and Res denotes the resource set of the database.

Given a query Q and its relaxation Q_i, we use the ratio between $Common(Q, Q_i)$ and the information content of Q to define the similarity value $Sim(Q, Q_i)$ as follows:

$$\begin{aligned} Sim(Q, Q_i) &= \frac{Common(Q, Q_i)}{I(Q)} \\ &= \frac{I(Q_i)}{I(Q)} = \frac{log\,\Pr(Q_i)}{log\,\Pr(Q)} \end{aligned} \qquad (1)$$

Since for Q, $\Pr(Q)$ is fixed, we also have:

$$\begin{aligned} Sim(Q, Q_i) &\propto -log\,\Pr(Q_i) \\ &\propto \frac{1}{\Pr(Q_i)} \\ &= \frac{1}{\frac{|Ans(Q_i)|}{|Res|}} \end{aligned} \qquad (2)$$

Given two relaxations Q_i and Q_j of query Q if $Sim(Q, Q_i) \geqslant Sim(Q, Q_j)$, results of Q_i are returned prior to Q_j. Note that if query Q has no answers, then $\Pr(Q) = 0$ and formula (1) would make no sense. In this case, for computing $Sim(Q, Q_i)$, we can use a constant $\Pr(Q) = \frac{1}{|Res|}$ to replace 0 in formula (1).

When relaxed query Q_i has no answers in the database,i.e. $\Pr(Q_i) = 0$, we have $Sim(Q, Q_i) = 0$ according to formula (1). This agrees with the fact that a relaxed query with no answers in the database is not what we want.

We also have propositions as follows:

Proposition 1.(Consistency) Given Q and its relaxation Q_i and Q_j, if $Q_i \prec Q_j$, then $Sim(Q, Q_i) \geqslant Sim(Q, Q_j)$.

Proof. According to formula (2), $Sim(Q, Q_i)$ and $Sim(Q, Q_j)$ are proportional to $\frac{1}{\Pr(Q_i)}$ and $\frac{1}{\Pr(Q_j)}$. Obviously, if $Q_i \prec Q_j$, $\Pr(Q_i) \leqslant \Pr(Q_j)$. Thus, we have $Sim(Q, Q_i) \geqslant Sim(Q, Q_j)$. □

Proposition 2. Given Q and its relaxation Q_i, if Q and Q_i are identical, then $Sim(Q, Q_i) = 1$.

From formula (2), we can see that the similarity between query Q and its relaxation Q_i is proportional to $\frac{1}{\Pr(Q_i)}$. However, computing $\Pr(Q_i)$ is not straightforward, we will discuss it in the next section.

4 Computing the Similarity efficiently

In this section, we will present the method for computing the similarities between the original query and its relaxations efficiently.

We have given the formula to compute the similarity value of relaxed queries. From formula (1), we can observe that for computing $Sim(Q, Q_i)$, we have to know $\Pr(Q_i)$, which is equal to $\frac{|Ans(Q_i)|}{|Res|}$. In the RDF database, $|Res|$ is the number of resources, which is a constant and easy to obtain. $|Ans(Q_i)|$ is the number of answers of Q_i. The naive way to compute $|Ans(Q_i)|$ is executing query Q_i in the database. However, it would not be time efficient because the number of possible relaxed queries is usually huge. Thus, we resort to approximate methods to compute $|Ans(Q_i)|$ efficiently without executing Q_i.

4.1 Selectivity estimation for star patterns using Bayesian networks

Selectivity estimation technology can be employed to estimate $|Ans(Q_i)|$. We use $sel(Q_i)$ to denote an estimation of $|Ans(Q_i)|$. Some work has been done on estimating the selectivity of RDF triple patterns. In [2, 11] the join uniformity assumption is made when estimating the joined triple patterns with bound subjects or objects (i.e., the subjects or objects are concrete values). It assumes that each triple satisfying a triple pattern is equally likely to join with the triples satisfying the other triple pattern. But this assumption does not fit the reality and would lead to great errors because it never considers the correlations among properties in a star query pattern. Thus we propose a method to estimate the selectivity more precisely with considering the correlations among properties.

For a group of subjects, choose a set of single-valued properties that often occur together. And we can construct a table R called cluster-property table for these properties. For example, Fig.4 shows a cluster-property table with three attributes *ResearchTopic*, *TeacherOf*, *ReviewerOf*. Each row of the table corresponds to a subject with values for the three properties.

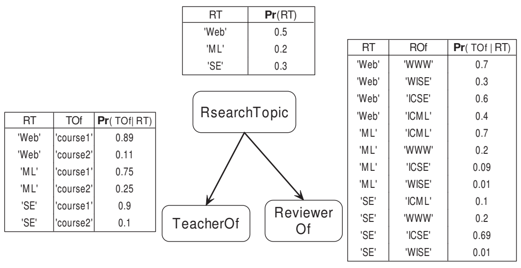
Given a star pattern Q with predicates $prop_1, prop_2, \cdots, prop_n$, the results of query Q should fall in the corresponding cluster-property table R. We denote the joint probability distribution over values of properties $prop_1$, $prop_2$, $\cdots$, $prop_n$ as $\Pr(prop_1 = o_1, prop_2 = o_2, \cdots, prop_n = o_n)$, where o_i is the value of property $prop_i$. We have:

$$sel(Q) = \Pr(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n) \cdot |T_R| \tag{3}$$

where $sel(Q)$ is the number of results of query Q and $|T_R|$ is the number of rows in the cluster-property table R.

Subject	ResearchTopic	ReviewerOf	TeacherOf
S_1	Web	WWW	course1
S_2	Web	WISE	course1
S_3	Web	WISE	course2
S_4	Web	WWW	course1
S_5	Software	ICSE	course2
S_6	Machine Learning	ICML	course3
...	....	....	....

Fig. 4. Cluster property table.



RT	**Pr**(RT)
'Web'	0.5
'ML'	0.2
'SE'	0.3

RT	TOf	**Pr**(TOf| RT)
'Web'	'course1'	0.89
'Web'	'course2'	0.11
'ML'	'course1'	0.75
'ML'	'course2'	0.25
'SE'	'course1'	0.9
'SE'	'course2'	0.1

RT	ROf	**Pr**(TOf | RT)
'Web'	'WWW'	0.7
'Web'	'WISE'	0.3
'Web'	'ICSE'	0.6
'Web'	'ICML'	0.4
'ML'	'ICML'	0.7
'ML'	'WWW'	0.2
'ML'	'ICSE'	0.09
'ML'	'WISE'	0.01
'SE'	'ICML'	0.1
'SE'	'WWW'	0.2
'SE'	'ICSE'	0.69
'SE'	'WISE'	0.01

Fig. 5. Bayesian Networks.

Notice that the possible combinations of values of properties would be exponential and it is impossible to explicitly store $\Pr(prop_1 = o_1, prop_2 = o_2, \cdots, prop_n = o_n)$. Thus, we need an appropriate structure to *approximately* store the joint probability distribution information. The Bayesian network [12] can approximately represent the probability distribution over a set of variables using a little space. Bayesian networks make use of Bayes' Rule and conditional independence assumption to compactly represent the full joint probability distribution. Let X, Y, Z be three discrete valued random variables. We say that X is conditionally independent of Y given Z if the probability distribution of X is independent of the value of Y given a value for Z; that is:

$$\Pr(X = x_i|Y = y_j, Z = z_k) = \Pr(X = x_i|Z = z_k)$$

where x_i, y_j, z_k are values of variables X, Y, Z. The conditional independence assumptions associated with a Bayesian network and conditional probability tables (CPTs), determine a joint probability distribution. For example, in Fig.5, a Bayesian network is constructed on cluster property table in Fig.4. We can see that properties *TeacherOf* and *ReviewerOf* are conditional independent given condition *ResearchTopic*, which means if we already know the research topic of some person, knowing his teaching information does not make any difference to our beliefs about his review information.

For a star query Q with properties $prop_1 = o_1$, $prop_2 = o_2$, $\cdots$, $prop_n = o_n$ and a Bayesian network β, we have:

$$\begin{aligned} &Pr_\beta(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n) \\ &= \prod_{i=1}^{n} \Pr(prop_i = o_i \mid Parents(prop_i) = \boldsymbol{o_k}) \end{aligned}$$

where $Parents(prop_i)$ denotes the set of immediate predecessors of $prop_i$ in the network β and $\boldsymbol{o_k}$ denotes the set of values of $Parents(prop_i)$. Note that for computing $\Pr(prop_i \mid parents(prop_i) = \boldsymbol{o_k})$, we only need to know the values of $prop_i$'s parent properties, which would save a lot of space in practice. So given the Bayesian network β, we can use $\Pr_\beta(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n)$ to *approximately* represent $\Pr(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n)$. We have:

$$\begin{aligned} sel(Q) &= \Pr(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n) \cdot |T_R| \\ &\approx Pr_\beta(prop_1 = o_1, prop_2 = o_2, ..., prop_n = o_n) \cdot |T_R| \\ &= \prod_{i=1}^{n} \Pr(prop_i = o_i \mid Parents(prop_i) = \boldsymbol{o_k}) \cdot |T_R| \end{aligned} \tag{4}$$

For example, given the star pattern Q(?x, ResearchTopic, 'Web')(?x, TeacherOf, 'course1') (?x, ReviewerOf, 'WWW') Bayesian network described in Fig.5, we compute the selectivity of Q as follows:

$$\begin{aligned} sel(Q) &= \Pr(ResearchTopic = \text{'Web'}, TeacherOf = \text{'course1'}, \\ &\quad ReviewerOf = \text{'WWW'}) \cdot |T_R| \\ &= \Pr(TeacherOf = \text{'course1'} \mid ResearchTopic = \text{'Web'}) \\ &\quad \cdot \Pr(ReviewerOf = \text{'WWW'} \mid ResearchTopic = \text{'Web'}) \\ &\quad \cdot \Pr(ResearchTopic = \text{'Web'}) \cdot |T_R| \\ &= 0.89 \cdot 0.7 \cdot 0.5 \cdot |T_R| = 0.3115 \cdot |T_R| \end{aligned}$$

It should be noted that relaxed queries may contain some properties which subsume properties in a property cluster table. Suppose that we have a property table R containing properties p_1,p_2,$\cdots p_n$ and we construct a Bayesian network on this table. Given a relaxed query Q (?x p o_1)(?x p_3 o_3)$\cdots$(?x p_n o_n), if property p subsumes p_1 and p_2, we estimate the selectivity of Q as follows:

$$\begin{aligned} sel(Q) &= \Pr(p = o_1, p_3 = o_3, ..., p_n = o_n) \cdot |T_R| \\ &\approx Pr_\beta(p_1 = o_1, p_3 = o_3, ..., p_n = o_n) \cdot |T_R| \\ &\quad + Pr_\beta(p_2 = o_1, p_3 = o_3, ..., p_n = o_n) \cdot |T_R| \\ &= \sum_{p_i \prec p} (\prod_{j \neq 2, \text{if } i=1; j \neq 1, \text{if } i=2}^{n} \Pr(p_j = o_j \mid Parents(p_j) = \boldsymbol{o_k})) \cdot |T_R| \end{aligned}$$

4.2 Learning Bayesian networks

To approximately represent the joint probability distribution of property values for selectivity estimation, we need to construct Bayesian networks for each cluster-property table. Before building Bayesian networks, the domain values are first discretized and clustered into equi-width subsets.

Bayesian network learning includes learning the structure of the DAG in a Bayesian network and parameters (i.e., conditional probability tables). A *score criterion* $score_\beta$ is a function that assigns a score to each possible DAG G' of the Bayesian network β under consideration based on the data.

$$score_\beta(d, G') = \prod_{i=1}^{n} \Pr(d, X_i, Parents(X_i)^{(G')})$$

where d is the data in the the cluster-property table and $Parents(X_i)^{(G')}$ is the set of parents of variable X_i in DAG G'. Bayesian network learning is to find a DAG that best fits the data (with the highest $score_\beta$), which is also called *model selection*. Much research has been conducted in Bayesian network learning [13, 14]. We use K2 [15] algorithm to learn the structure of Bayesian networks. It adopts a greedy search that maximizes $score_\beta(d, G')$ approximately. For each variable X_i, it locally finds a value of $Parents(X_i)$ that maximizes $score_\beta(d, X_i, Parents(X_i))$. This method needs to provide the temporal ordering of variables, which can significantly reduce the search space. In our implementation, we input the temporal ordering of variables by hand.

5 Top-K query processing

In this section, we discuss how to acquire top-k approximate answers through query relaxation. Fig.6 shows a relaxation graph of query Q. From the relaxation graph of query Q, we can see that there are 9 relaxed queries of Q and $Q_1 \prec Q_{11}$, $Q_1 \prec Q_{12} \prec Q_{121}$, $Q_2 \prec Q_{21}$, $Q_3 \prec Q_{31}$, $Q_3 \prec Q_{32}$. In the relaxation graph, there exists an edge from relaxed queries Q_i to Q_j if Q_i is directly subsumed by Q_j(i.e., $Q_i \prec Q_j$ and $\nexists Q_m$ s.t. $Q_i \prec Q_m \prec Q_j$). We also call Q_j is a child of Q_i in the relaxation graph.

Suppose query Q has no answers returned and we want to acquire top-k approximate answers of Q. The relaxation process for obtaining top-k approximate answers is implemented by the best-first search. We first add the children of Q to the *Candidates* set, which are promising to be the best relaxation of query Q. Then select the best candidate Q' (with the highest similarity value) in the *Candidates* set to execute and add the children of Q' to the *Candidates* set. Repeat this process until we get enough answers. This process is described in Algorithm 1-*topkAlgorithm*.

.

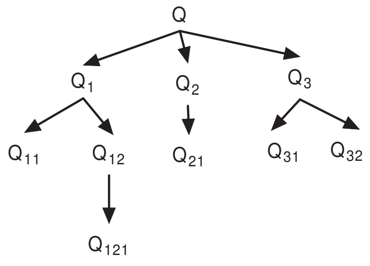


Fig. 6. The Relaxation Graph of Query Q

Algorithm 1 *topkAlgorithm*

Input: Query Q; k(the number of answers required);
Output: top-k approximate answers of query Q;

```
1: $Answers = \Omega$; $Candidates = \Omega$;
2: Add $Q$'s child nodes into $Candidates$;
3: while $|Answers| < k$ and $|Candidates| <> \Omega$ do
4:     Select the $Q'$ with the best similarity value from $Candidates$;
5:     Add child nodes of $Q'$ to $Candidates$
6:     Add answers of $Q'$ to $Answers$
7:     $Candidates = Candidates \setminus Q'$
8: return $Answers$;
```

6 Experiments

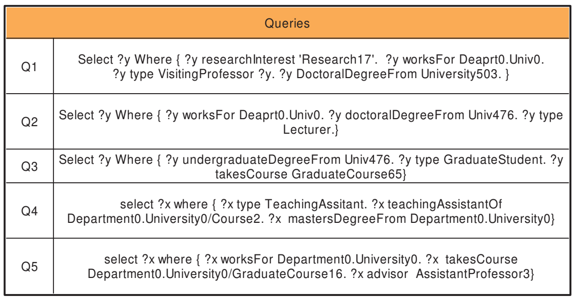
In this section, we conduct experiments to verify our methods.

Experiment setup. The data is stored in and managed by Mysql 5.0. All algorithms are implemented using Jena (http://jena.sourceforge.net/). We run all algorithms on a windows XP professional system with P4 3G CPU and 2 GB RAM.

Data sets. We construct the dataset using the Lehigh University Benchmark LUBM [3]. LUBM consists of a university domain ontology containing 32 properties and 43 classes. The dataset used in the experiments contains 600k triples.

Query Load. We develop 5 star queries shown in Fig.7.

Bayesian Networks. We first construct Bayesian Networks from the dataset for estimating the selectivity of relaxed queries. Fig.8 shows one of Bayesian Networks Learned from the dataset for "Professor" entity. We can see that "Professor" entity has 6 properties. Properties "MastersDegreeFrom"and "DoctoralDegreeFrom"are independent with other properties. The values of property "ResearchInterest"depends on the values of properties "type" and "worksFor". According to this Bayesian Network, we can easily estimate the selectivity of star queries on "Professor" entity.



Queries	
Q1	Select ?y Where { ?y researchInterest 'Research17'. ?y worksFor Deaprt0.Univ0. ?y type VisitingProfessor ?y. ?y DoctoralDegreeFrom University503. }
Q2	Select ?y Where { ?y worksFor Deaprt0.Univ0. ?y doctoralDegreeFrom Univ476. ?y type Lecturer.}
Q3	Select ?y Where { ?y undergraduateDegreeFrom Univ476. ?y type GraduateStudent. ?y takesCourse GraduateCourse65}
Q4	select ?x where { ?x type TeachingAssitant. ?x teachingAssistantOf Department0.University0/Course2. ?x mastersDegreeFrom Department0.University0}
Q5	select ?x where { ?x worksFor Department0.University0. ?x takesCourse Department0.University0/GraduateCourse16. ?x advisor AssistantProfessor3}

Fig. 7. Queries Used In the Experiments

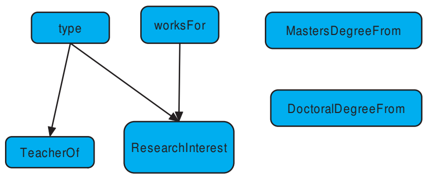


Fig. 8. One of Bayesian Networks learned from the dataset for "Professor"entity

Performance. We conduct experiments to evaluate the performance of our relaxation algorithm. We first fix the number of approximate answers K=50. Fig.9 (a) and (b) show the running time and the number of relaxed queries executed for the relaxation algorithm. The similarity values of the approximate answers of Q_1 to Q_5 are $\geq$0.61, $\geq$0.67, $\geq$0.79, $\geq$0.71, $\geq$0.69, respectively. Our approach can avoid some unnecessary relaxations. For example, query Q_1 has four triple patterns. There are many ways to relax Q_1, but some of them have no answers. From the Bayesian network constructed (shown in Fig.8), we know the data distribution in the database, which indicates that there is no visiting professor who works for "Department0.University0"and is interested in "Research 17". Thus, there will be no answers by relaxing triple pattern(?x DoctoralDegreeFrom University503) to (?x DoctoralDegreeFrom ?w). We relax Q_1 through replacing "VisitingProfessor" to "Professor"(super class of"VisitingProfessor"), which is close to the original query and avoids the empty answer set.

When K =150, from Fig.9 (c) and (d), we can see that the running time and the number of relaxed queries executed increase. On the contrary, the similarity values of the approximate answers of Q_1 to Q_5 are $\geq$0.57, $\geq$0.64, $\geq$0.69, $\geq$0.66, $\geq$0.63, respectively. Our relaxation algorithm returns approximate answers incrementally and users can stop the relaxation process at any time when they are satisfied with the answers generated.

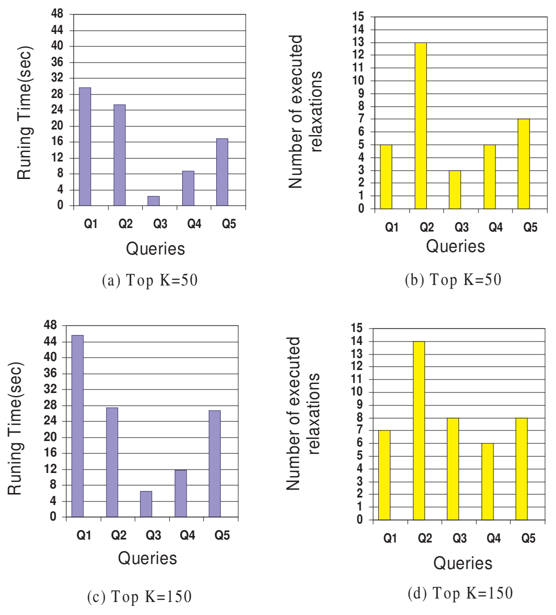


Fig. 9. The performance of our relaxation algorithm

7 Related Work

Hurtado *et al.* [1] proposed the RDF query relaxation method through RDF(s) entailment and relax the original user query using the ontology information. They also proposed the ranking model to rank the relaxed queries according to the subsume relationship among them. However, for some relaxed queries, if there does not exit subsume relationship among them, it fails to rank them.

Similarity measures for concepts have been developed in the previous decades. Generally, these methods can be categorized into three groups: edge counting-based methods [6], information theory-based methods [4]. For evaluating the similarity of queries, in [5], the authors proposed the ranking model using distance based measure on RDFs ontology. However, this measure is only based on ontology information without considering the real data distribution in the database, it is possible that relaxed queries ranked high still have no answers.

Some related work has been done, such as query relaxation on XML data [9, 8, 7]. Amer-Yahia *et al.* [10] presented a method for computing 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 applied to query relaxation on RDF data. Cooperative query answering [16] is 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.

8 Conclusion

Query relaxation is important for querying RDF data flexibly. In this paper, we presented a ranking model which considers the real data distribution in the RDF database to rank the relaxed queries. The relaxations are ranked based on their similarity values to the original query. The similarity values of relaxed star queries are computed efficiently using Bayesian networks. We also presented the algorithm to answer top-k queries according to our ranking model.

Acknowledgments

This work was partially supported by the Australian Research Council Discovery Project under the grant number DP0878405.

References

1. C. A. Hurtado, Al Poulovassilis, Peter T. Wood.: A Relaxed Approach to RDF Querying. In ISWC, pp. 314-328 (2006)
2. M. Stocker, A. Seaborne, Abraham Bernstein, Christoph Kiefer, Dave Reynolds: SPARQL basic graph pattern optimization using selectivity estimation. In WWW, pages:595-604, 2008.
3. Y. Guo, Z. Pan, J. Heflin.: An Evaluation of Knowledge Base Systems for Large OWL Datasets. In ISWC, page:274-288, 2004.
4. P. Resnik: Using Information Content to Evaluate Semantic Similarity in a Taxonomy. IJCAI page: 448-453,1995.
5. H. Huang, C. Liu, X. Zhou: Computing Relaxed Answers on RDF Databases. WISE 2008: 163-175
6. L. J. Ho: Information Retrieval Based on Conceptual Distance in Is-A Hierarchies. Journal of Documentation, v49 n2 p188-207 Jun 1993.
7. N. Fuhr, K. Grobjohann: XIRQL: A Query Language for Information Retrieval in XML Documents. SIGIR 2001: 172-180
8. Query Relaxation for XML Model. PhD thesis, Department of Computer Science, University of Califonia Los angeles, 2002.
9. Y. Kanza, Y. Sagiv: Flexible Queries Over Semistructured Data. PODS 2001
10. S. A.-Yahia, S. Cho, D. Srivastava.: Tree Pattern Relaxation. In: EDBT , pp. 496-513 (2002)
11. T. Neumann, G. Weikum: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1): 647-659, 2008.
12. Pearl, J.: Probabilistic reasoning in intelligent systems: Networks of plausible inference. San Mateo, CA: Morgan-Kaufmann, 1988.
13. Wray L. Buntine: Operations for Learning with Graphical Models. In J. Artif. Intell. Res. (JAIR) 2: 159-225,1994.
14. D. Heckerman, D. Geiger, D. Maxwell Chickering: Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. In Machine Learning 20(3): 197-243, 1995.
15. G. F. Cooper, E. Herskovits: A Bayesian Method for the Induction of Probabilistic Networks from Data. In Machine Learning 9: 309-347, 1992.
16. J. M. Kleinberg.: Authoritative Sources in a Hyperlinked Environment. J. ACM 46(5): 604-632 (1999)