A Relaxed Approach to RDF Querying

Carlos A. Hurtado*[1], Alexandra Poulovassilis[2], and Peter T. Wood[2]

[1] Universidad de Chile (`churtado@dcc.uchile.cl`)
[2] Birkbeck, University of London ({`ap,ptw`}`@dcs.bbk.ac.uk`)

Abstract. We explore flexible querying of RDF data, with the aim of making it possible to return data satisfying query conditions with varying degrees of exactness, and also to rank the results of a query depending on how "closely" they satisfy the query conditions. We make queries more flexible by logical relaxation of their conditions based on RDFS entailment and RDFS ontologies. We develop a notion of ranking of query answers, and present a query processing algorithm for incrementally computing the relaxed answer of a query. Our approach has application in scenarios where there is a lack of understanding of the ontology underlying the data, or where the data objects have heterogeneous sets of properties or irregular structures.

1 Introduction

The conjunctive fragment of most RDF query languages (e.g., see [8, 9]) consists of queries of the form $H \leftarrow B$, where the body of the query B is a graph pattern, that is, an RDF graph over IRIs, literals, blanks, and variables. The head of the query H is either a graph pattern or a tuple variable (list of variables). The semantics of these queries is simple. It is based on finding matchings from the body of the query to the data and then applying the matchings to the head of the query to obtain the answers.

Recently, the W3C RDF data access group has emphasized the importance of enhancing RDF query languages to meet the requirements of contexts where RDF can be used to solve real problems. In particular, it has been stated that in RDF querying "it must be possible to express a query that does not fail when some specified part of the query fails to match" [5]. This requirement has motivated the `OPTIONAL` clause, presented in the emerging SPARQL W3C proposal for querying RDF [13] and previously introduced in SeRQL [3]. The `OPTIONAL` clause allows the query to find matchings that fail to match some conditions in the body. In contrast to other approaches to flexible querying (e.g., [1, 11]), the `OPTIONAL` construct incorporates flexibility from a "logical" standpoint, via relaxation of the query's conditions. This idea, however, is exploited only to a limited extent, since the conditions of a query could be relaxed in ways other than simply dropping optional triple patterns, for example by replacing constants with variables or by using the class and property hierarchies in an ontology associated with the data (such as that shown in Figure 1).

* Carlos A. Hurtado was supported by Millennium Nucleus, Center for Web Research (P04-067-F), Mideplan, and by project FONDECYT 1030810, Chile.

1.1 RDFS Ontologies

It is common that users interact with RDF applications in the context of an ontology. We assume that the ontology is modeled as an RDF graph with interpreted RDFS vocabulary. The RDFS vocabulary defines classes and properties that may be used for describing groups of related resources and relationships between resources. To state that a resource is an instance of a class, the property rdf:type may be used. In this paper we use a fragment of the RDFS vocabulary, which comprises (in brackets is the shorter name we will use) rdfs: range [`range`], rdfs:domain [`dom`], rdf:type [`type`], rdfs: subClassOf [`sc`] and rdfs:subPropertyOf [`sp`][3].

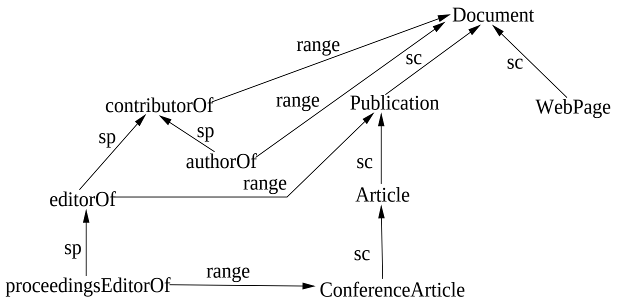

Fig. 1. An RDFS ontology modeling documents and people who contribute to them.

As an example, the ontology of Figure 1 is used to model documents along with properties that model different ways people contribute to them (e.g., as authors, editors, or being the editor of the proceedings where an article is published).

1.2 The `RELAX` Clause

In this paper, we propose the introduction of a `RELAX` clause as a generalization of the `OPTIONAL` clause for conjunctive queries. As an example, consider the following SPARQL-like query Q[4]:

$$?Z, ?Y \leftarrow \{(?X, name, ?Z), \texttt{OPTIONAL}\{(?X, proceedingsEditorOf, ?Y)\}\}.$$

The body of this query is a graph pattern comprising two triple patterns. This query returns names of people along with the IRIs of conference articles whose

[3] We omit in this paper vocabulary used to refer to basic classes in RDF/S such as rdf: Property, rdfs: Class, rdfs:Resource, rdfs:Literal, rdfs:XMLLiteral, rdfs:Datatype, among others. We also omit vocabulary for lists, collections, and variations on these, as well as vocabulary used to place comments in RDF/S data.

[4] SPARQL has SQL-like syntax; for brevity, in this paper we express queries as rules.

proceedings they have edited. Because the second triple pattern in the body of the query is within the scope of an `OPTIONAL` clause, the query also returns names of people for which the second pattern fails to match the data (i.e., people who have not edited proceedings).

Now consider the ontology of Figure 1. Although the user may want to retrieve editors of proceedings at first, she/he might also be interested in knowing about people who have contributed to publications in other roles, along with the publications themselves. In order to save the user the effort of inspecting the ontology and rewriting the query, the system could automatically return more relaxed answers for the same original query. This is achieved by rewriting Q to replace `OPTIONAL` with `RELAX`. Now after returning editors of conference proceedings, the system can replace the triple pattern $(?X, \mathit{proceedingsEditorOf}, ?Y)$ with $(?X, \mathit{editorOf}, ?Y)$, yielding a new, relaxed query that returns editors of publications along with their publications. Subsequently, this triple pattern can be rewritten to the triple pattern $(?X, \mathit{contributorOf}, ?Y)$ to obtain more general answers.

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

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

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

(Simple Entailment) (7) For a map $\mu : G' \to G : \frac{G}{G'}$

Fig. 2. RDFS Inference Rules

The idea of making queries more flexible by the logical relaxation of their conditions is not new in database research. Gaasterland et al. [7] established the foundations of such a mechanism in the context of deductive databases and logic programming, and called the technique *query relaxation*.

1.3 Notion of Query Relaxation for RDF

We study the query relaxation problem in the setting of the RDF/S data model and RDF query languages and show that query relaxation can be naturally formalized using RDFS entailment. We use an operational semantics for the notion of RDFS entailment, denoted $\models$, characterized by the derivation rules given in Figure 2 (for details, see [8, 10]). Rules in groups (A), (B), and (C) describe the semantics of the RDFS vocabulary we use in this paper (i.e., `sp`, `sc`, `type`, `dom`, and `range`), and rule 7 (which is based on the notion of map which we will explain in Section 2), essentially states that blank nodes behave like existentially quantified variables. As an example, from a graph we can entail another graph which replaces constants with blanks or blanks with other blanks.

Intuitively, as RDFS entailment is characterized by the rules of Figure 2, a relaxed triple pattern t' can be obtained from triple t by applying the derivation rules to t and triples from the ontology. As an example, the triple pattern $(?X, \mathit{proceedingsEditorOf}, ?Y)$ can be relaxed to $(?X, \mathit{editorOf}, ?Y)$, by applying rule 3 to the former and the triple $(\mathit{proceedingsEditorOf}, \texttt{sp}, \mathit{editorOf})$ in the ontology of Figure 1. The different relaxed versions of an original query are obtained by combining relaxations of triple patterns that appear inside a `RELAX` clause.

The notion of query relaxation we propose naturally subsumes two broad classes of relaxations (further types of relaxations within these two classes are listed in Section 3.4). The first class of relaxation, which we call *simple relaxations*, consists of relaxations that can be entailed without an ontology, which include dropping triple patterns, replacing constants with variables, and breaking join dependencies. These are captured by derivation rule 7 (Figure 2). The second class of relaxations, which we call *ontology relaxations*, includes relaxations entailed using information from the ontology and are captured by rule groups (A),(B) and (C); these include relaxing type conditions, relaxing properties using domain or range restrictions and others.

1.4 Summary of Contributions and Outline

In this paper, we develop a framework for query relaxation for RDF. We introduce a notion of query relaxation based on RDFS entailment, which naturally incorporates RDFS ontologies and captures necessary information for relaxation such as the class and property hierarchies.

By formalizing query relaxation in terms of entailment, we obtain a semantic notion which is by no means limited to RDFS and could also be extended to more expressive settings such as OWL entailment and OWL ontologies, to capture further relaxations. Our framework generalizes, for the conjunctive fragment of SPARQL, the idea of dropping query conditions provided by the `OPTIONAL` construct.

An essential aspect of our proposal, which sets it apart from previous work on query relaxation, is to rank the results of a query based on how “closely” they satisfy the query. We present a notion of ranking based on a structure called the *relaxation graph*, in which relaxed versions of the original query are ordered from less to more general from a logical standpoint. Since the relaxation graph is based on logic subsumption, ranking does not depend on any syntactic condition on the knowledge used for relaxation (such as rule ordering in logic-programming approaches [7]). Finally, we sketch a query processing algorithm to compute the relaxed answer of a query, and examine its correctness and complexity.

The rest of the paper is organized as follows. Section 2 introduces preliminary notation. Section 3 formalizes query relaxation and Section 4 studies query processing. In Section 5 we study related work and in Section 6 we present some concluding remarks.

2 Preliminary Definitions

In this section we present the basic notation and definitions that will be used subsequently in this paper. Some of these were introduced in [2, 8, 10, 12].

RDF Graphs In this paper we work with RDF graphs which may mention the RDFS vocabulary. We assume there are infinite sets I (IRIs), B (blank nodes), and L (RDF literals). The elements in $I \cup B \cup L$ are called RDF *terms*. A triple $(v_1, v_2, v_3) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an *RDF triple*. In such a triple, v_1 is called the *subject*, v_2 the *predicate* and v_3 the *object*. An *RDF graph* (just graph from now on) is a set of RDF triples. Given two RDF graphs G_1, G_2, a *map* from G_1 to G_2 is a function μ from terms of G_1 to terms of G_2, preserving IRIs and literals, such that for each triple $(a, b, c) \in G_1$ we have $(\mu(a), \mu(b), \mu(c))) \in G_2$.

Entailment We will decompose RDFS entailment into two notions of entailment. The first is simple entailment [10], which depends only on the basic logical form of RDF graphs and therefore holds for any vocabulary. An RDF graph G_1 *simply entails* G_2, denoted $G_1 \models_{\texttt{simple}} G_2$, if and only if there exists a map from G_2 to G_1. That is, simple entailment is captured by rule 7 of Figure 2.

The second notion of entailment captures the semantics added by the RDFS vocabulary. We write that $G_1 \models_{\texttt{rule}} G_2$ if G_2 can be derived from G_1 by iteratively applying rules in groups (A), (B) and (C) of Figure 2. In this paper, we also use a notion of closure of an RDF graph G [10], denoted $\mathrm{cl}(G)$, which is the closure of G under the rules in groups (A), (B) and (C). We have that $G_1 \models_{\texttt{rule}} G_2$ if and only if $G_2 \in \mathrm{cl}(G_1)$.

Now, by a result from from [10], RDFS entailment (for the fragment of RDFS we use in this paper) can be characterized as follows: G_1 RDFS-entails G_2, denoted $G_1 \models_{\texttt{RDFS}} G_2$, if and only if there is a graph G such that $G_1 \models_{\texttt{rule}} G$ and $G \models_{\texttt{simple}} G_2$. An alternative characterization of RDFS entailment is the following: $G_1 \models_{\texttt{RDFS}} G_2$ if and only if there is a map from G_2 to $\mathrm{cl}(G_1)$. Therefore, in order to test the entailment $G_1 \models_{\texttt{RDFS}} G_2$, we can first apply rules in groups (A), (B), and (C) to compute $\mathrm{cl}(G_1)$, and then check whether there exists a map from G_2 to $\mathrm{cl}(G_1)$.

Graph Patterns Consider a set of variables V disjoint from the sets I, B, and L. A *triple pattern* is a triple $(v_1, v_2, v_3) \in (I \cup V) \times (I \cup V) \times (I \cup V \cup L)$. A *graph pattern* is a set of triple patterns. Given a graph pattern P, we denote by $\texttt{var}(P)$ the variables mentioned in P. The following notation is needed to define triple pattern relaxation in Section 3. The notion of map is generalized to graph patterns by treating variables as blank nodes. . In addition, t_1 is S-isomorphic to t_2 if there are maps μ_1 from t_1 to t_2 and μ_2 from t_2 to t_1 that both preserve S. In our examples, variables are indicated by a leading question mark, while literals are enclosed in quotes.

Conjunctive Queries for RDF A *conjunctive query* Q is an expression $T \leftarrow B$, where B is a graph pattern, and $T = \langle T_1, \ldots, T_n \rangle$ is a list of variables which belongs to $\texttt{var}(B)$. (The framework formalized in this paper can be easily extended to queries with graph patterns as query heads.) We denote T by $\texttt{Head}(Q)$, and B by $\texttt{Body}(Q)$. A query Q may be formulated over an RDFS ontology O, which means that Q may mention vocabulary from O and its answer is obtained taking into account the semantics of O. We assume that the ontology is well designed in the sense that predicates of triples in O cannot be in the set $\{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$. We define a *matching* to be a function from variables in $\texttt{Body}(Q)$ to blanks, IRIs and literals. Given a matching Θ, we denote by $\Theta(\texttt{Body}(Q))$ the graph resulting from $\texttt{Body}(Q)$ by replacing each variable X by $\Theta(X)$. Given an RDF graph G, the *answer* of Q is the set of tuples, denoted $\texttt{ans}(Q, O, G)$, defined as follows: for each matching Θ such that $\Theta(\texttt{Body}(Q)) \subseteq \mathrm{cl}(O \cup G)$, return $\Theta(\texttt{Head}(Q))$. When O is clear from the context, we omit it, and write $\texttt{ans}(Q, G)$ instead of $\texttt{ans}(Q, O, G)$.

3 Formalizing Query Relaxation

We will present a relaxed semantics for queries in a stepwise manner. In Section 3.1, we present the notion of relaxation of triple patterns, and in Section 3.2 we introduce the notion of the relaxation graph of a triple pattern. This is used in Section 3.3 to define the relaxation graph of a query. The relaxation graph is the basis for the notion of the relaxed answer and ranking of a query we propose in Section 3.5. In Section 3.4, we explain different types of relaxations.

3.1 Triple Pattern Relaxation

We model relaxation as a combination of two types of relaxations, *ontology relaxation* and *simple relaxation*. Intuitively, the former comprises relaxations that are based on the ontology at hand and do not replace terms of the original triple pattern. In contrast, simple relaxations consist only of replacements of terms of the original triple pattern (e.g., replacing a literal or URI with a variable or a variable with another variable).

Relaxation will be defined in the context of an ontology, denoted by O, and a set of variables, called *fixed variables*, denoted by F. So we fix O and F for the definitions that follow.

Let t_1, t_2 be triple patterns, where $t_1 \notin \mathrm{cl}(O)$, $t_2 \notin \mathrm{cl}(O)$, and $\texttt{var}(t_2) = \texttt{var}(t_1) \subseteq F$. Ontology relaxation is defined as follows: $t_1 \prec^*_{\texttt{onto}} t_2$ if $(\{t_1\} \cup O) \models_{\texttt{rule}} t_2$. As an example, let O be the ontology of Figure 1 and let $F = \{?X\}$. Then, we have that $(?X, \texttt{type}, \mathit{ConferenceArticle}) \prec^*_{\texttt{onto}} (?X, \texttt{type}, \mathit{Article})$, and we have that $(\mathit{JohnRobert}, \mathit{ContributorOf}, ?X) \prec^*_{\texttt{onto}} (?X, \texttt{type}, \mathit{Document})$, among other ontology relaxations. It is not the case that $(?X, \mathit{ContributorOf}, ?Y) \prec^*_{\texttt{onto}} (?Y, \texttt{type}, \mathit{Document})$, since the set of variables of the triples are different.

Simple relaxation is defined as follows: $t_1 \prec^*_{\texttt{simple}} t_2$ if $t_1 \models_{\texttt{simple}} t_2$ via a map μ that preserves F (recall the notion of a map preserving a set of variables from Section 2). As an example, we have $(?X, \texttt{type}, \mathit{Article}) \prec^*_{\texttt{simple}} (?X, \texttt{type}, ?Z)$

and $(?X, \texttt{type}, \mathit{Article}) \prec^*_{\texttt{simple}} (?X, ?W, \mathit{Article})$, among other simple relaxations.

We now define relaxation. We say that t_2 *relaxes* t_1, denoted $t_1 \prec^* t_2$, if one of the following hold: (i) $t_1 \prec^*_{\texttt{onto}} t_2$, (ii) $t_1 \prec^*_{\texttt{simple}} t_2$, or (iii) there exists a triple pattern t such that $t_1 \prec^* t$ and $t \prec^* t_2$. The following proposition proves that simple relaxations always arise after ontology relaxations.

Proposition 1. *Let t_1, t_2 be triple patterns. Then $t_1 \prec^* t_2$ if and only if there exists t such that $t_1 \prec^*_{\texttt{onto}} t$ and $t \prec^*_{\texttt{simple}} t_2$.*

We will end this section by proving some properties of the relaxation relationships introduced. We define an ontology O to be acyclic if the subgraphs defined by `sc` and `sp` are acyclic. Acyclicity is considered good practice in modeling ontologies. Recall the notion of graph pattern isomorphism with respect to a set of fixed variables from Section 2.

Proposition 2. *Let $\prec^*_{\texttt{onto}}$, $\prec^*_{\texttt{simple}}$ and $\prec^*$ be defined in the context of an ontology O and a set F of fixed variables. (i) $\prec^*_{\texttt{onto}}$ is a partial order if and only if O is acyclic. (ii) $\prec^*_{\texttt{simple}}$ is a partial order up to F-isomorphism. (iii) $\prec^*$ is a partial order up to F-isomorphism if and only if O is acyclic.*

In what follows we assume that O is acyclic, and assume triple patterns to be equal if they are F-isomorphic. Therefore, we consider the relaxation relations to be partial orders. In particular, if a variable is not in F, without loss of generality we assume it appears in no more that one triple pattern. We denote by $\prec$ (direct relaxation) the reflexive and transitive reduction of $\prec^*$ (relaxation). We use similar notation for ontology and simple relaxation.

3.2 Relaxation Graph of a Triple Pattern

We are interested in relaxing each of the triple patterns that occurs inside the `RELAX` clause of a query, so we next adapt the relaxation relationship to use relaxation "above" a given triple pattern. The relaxation relation "above" a triple pattern t, denoted by $\prec^*_t$, is $\prec^*$ restricted to triple patterns t' such that $t \prec^* t'$, and where $F = \texttt{var}(t)$ (i.e., the variables of t are the fixed variables in the relaxation). The *relaxation graph* of a triple pattern t is the directed acyclic graph induced by $\prec_t$.

As an example, consider the ontology O of Figure 1. Let t be the triple pattern $(?X, \texttt{type}, \mathit{Publication})$. Figure 3 (A) shows the relaxation graph of $(?X, \texttt{type}, \mathit{Publication})$. We have that $?X$ is the unique fixed variable. The non-fixed variables in this graph are $?V1, \ldots, ?V5$. Figure 3 (B) shows the relaxation graph of $(\mathit{JohnRobert}, \mathit{editorOf}, ?X)$. Now the non-fixed variables are $?U1, \ldots, ?U9$. Notice that this pattern directly relaxes to $(?X, \texttt{type}, \mathit{Publication})$, so this relaxation graph has as a subgraph the relaxation graph of Figure 3 (A).

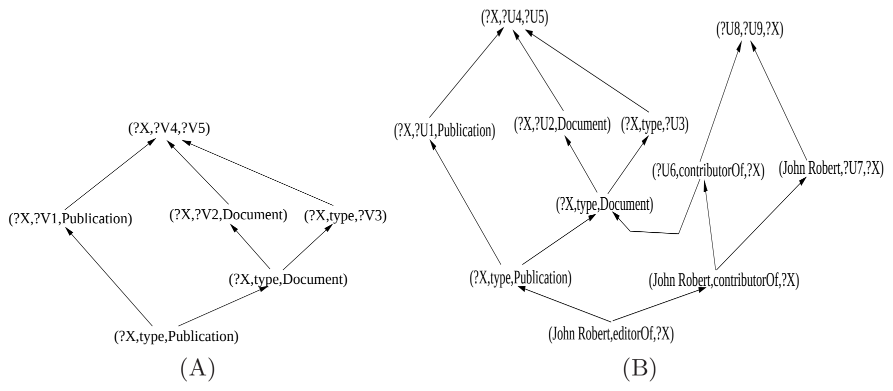

Fig. 3. (A) Relaxation graph of the triple pattern $(?X, \texttt{type}, \textit{Publication})$. (B) Relaxation graph of the triple pattern $(\textit{JohnRobert}, \textit{editorOf}, ?X)$.

3.3 Query Relaxation

In this section, we define query relaxation as the direct product of the relaxation relations of its triple patterns. We define the *direct product* of two partial order relations α_1, α_2, denoted $\alpha_1 \otimes \alpha_2$, as the relation α such that $(e_1, e_2)\ \alpha\ (e'_1, e'_2)$ if and only if $e_1\ \alpha_1\ e'_1$ and $e_2\ \alpha_2\ e'_2$. The generalization of this definition to more than two relations is straightforward.

Given a query Q, let $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$. For any triple t_i not inside a `RELAX` clause, we overload the notation $\prec^*_{t_i}$ and assume that t_i relaxes only to t_i. The relaxation relation "above" Q, denoted by $\prec^*_Q$, is defined as $\prec^*_{t_1} \otimes \prec^*_{t_2} \ldots \otimes \prec^*_{t_n}$. Direct relaxation, denoted $\prec_Q$, is the reflexive and transitive reduction of $\prec^*_Q$. The *relaxation graph* of Q is the directed acyclic graph induced by $\prec_Q$.

Each node $(t'_1, \ldots, t'_n)$ in the relaxation graph of Q denotes the conjunctive query $\texttt{Head}(Q) \leftarrow t'_1, \ldots, t'_n$. In order to avoid name clashes, we assume that the sets of non-fixed variables introduced in the relaxation relations of the triple pattern are pairwise disjoint.

3.4 Types of Relaxation

The notion of relaxation that we propose in this paper encompasses several different types of relaxation. Those captured by simple relaxation are as follows:

1. Dropping triple patterns. We can model the dropping of triple patterns by introducing an "empty" triple pattern, which can be regarded as a "true" condition to which any triple pattern relaxes. In this form, relaxation generalizes the use of the `OPTIONAL` clause within the conjunctive fragment of SPARQL.
2. Constant relaxation: replacing a constant with a variable in a triple pattern. This can be further classified according to whether the variable replaces a property or a subject/object constant.

3. Breaking join dependencies: generating new variable names for a variable that appears in multiple triple patterns. In order to model this type of relaxation, we first transform queries by applying variable substitution. If a variable $?X$ appears $n > 1$ times in a query Q we replace each occurrence with a different variable and add triple patterns $(?X_i, \texttt{equal}, ?X_j)$ for each pair of new variables $?X_i, ?X_j$ introduced. The predicate `equal` represents equality. Each of the equality clauses in a query can now also be subject to relaxation.

The following types of relaxation are captured by our notion of ontology relaxation (the examples given use the ontology of Figure 1):

1. Type relaxation: replacing a triple pattern $(a, \texttt{type}, b)$ with $(a, \texttt{type}, c)$, where $(b, \texttt{sc}, c) \in \mathrm{cl}(O)$. For example, the triple pattern $(?X, \texttt{type}, \mathit{ConferenceArticle})$ can be relaxed to $(?X, \texttt{type}, \mathit{Article})$ and then to $(?X, \texttt{type}, \mathit{Publication})$.
2. Predicate relaxation: replacing a triple pattern (a, p, b) with (a, q, c), where $(p, \texttt{sp}, q) \in \mathrm{cl}(O)$. For example, the triple pattern $(?X, \mathit{proceedingsEditorOf}, ?Y)$ can be relaxed to $(?X, \mathit{editorOf}, ?Y)$ and then to $(?X, \mathit{contributorOf}, ?Y)$.
3. Predicate to domain relaxation: replacing a triple pattern (a, p, b) with $(a, \texttt{type}, c)$, where $(p, \texttt{dom}, c) \in \mathrm{cl}(O)$. There are no domain declarations in Figure 1.
4. Predicate to range relaxation: replacing a triple pattern (a, p, b) with $(b, \texttt{type}, c)$, where $(p, \texttt{range}, c) \in \mathrm{cl}(O)$. For example, the triple pattern $(?X, \mathit{editorOf}, ?Y)$ can be relaxed to $(?Y, \texttt{type}, \mathit{Publication})$.
5. Additional relaxations induced by additional rules from Figure 2. Combinations of rules yield additional forms of relaxation. For example, the triple pattern $(\mathit{Article}, \texttt{sc}, ?Y)$ can be relaxed to $(\mathit{ConferenceArticle}, \texttt{sc}, ?Y)$.

3.5 Relaxed Answer and Ranking

Any algorithm that computes a relaxed answer to a query should also return the tuples in the relaxed answer according to some ordering. So the output of a query processing algorithm can be viewed as twofold: (a) the relaxed answer (already defined) and (b) a rank function that defines an ordering for the tuples in the relaxed answer. We next define the notions of *relaxed answer*, *rank function* and *consistency* of a rank function. Roughly, consistency means that the tuple ordering defined by the rank function agrees with the ordering of queries imposed by the query relaxation graph.

Let Q be an RDF query and G be an RDF graph. The *level of a query* Q_i in the relaxation graph of Q is the length of the shortest path from Q to Q_i. We denote by $\texttt{relax}(Q, k)$ the set of queries in the relaxation graph whose level is less than or equal to k. The *relaxed answer* of Q over G at level $k \geq 1$, $\texttt{ans}_{\texttt{relax}}(Q, G, k)$, is the set of tuples $\bigcup_{Q' \in \texttt{relax}(Q,k)} \texttt{ans}(Q', G)$. We will frequently mention $\texttt{ans}_{\texttt{relax}}(Q, G, k)$ in a context where k is fixed, and in this context we will write $\texttt{ans}_{\texttt{relax}}(Q, G, k)$ simply as $\texttt{ans}_{\texttt{relax}}(Q, G)$.

For a query Q' in the relaxation graph of Q, and an RDF graph G, we define $\texttt{newAnswer}(Q', G)$ as $\texttt{ans}(Q', G) - (\bigcup_{Q_i : Q_i \prec_Q^* Q'} \texttt{ans}(Q_i, G))$.

Let Q be a query and G be an RDF graph. A *rank function* for the relaxed answer of Q over G is any function $\tau_{Q,G}$ with signature $\tau_{Q,G} : (\mathtt{ans}_{\mathtt{relax}}(Q, G)) \rightarrow N$. A rank function $\tau_{Q,G}$ is *consistent* if and only if for each pair of tuples $t_i, t_j \in \mathtt{ans}_{\mathtt{relax}}(Q, G)$, if there are queries Q_i, Q_j such that $Q_i \prec^*_Q Q_j$, $t_i \in \mathtt{newAnswer}(Q_i, G)$, $t_j \in \mathtt{newAnswer}(Q_j, G)$, then $\tau_{Q,G}(t_i) < \tau_{Q,G}(t_j)$.

Notice that in consistent rank functions, tuples in $\mathtt{ans}(Q, G)$ are returned first among the tuples in the relaxed answer. The notion of answer ranking sketched here can be improved in several directions. We may impose additional ordering constraints in the relaxation graph. Other extensions may consider distance metrics based on paths in the relaxation graph.

4 Query Processing

In this section, we study the problem of computing the relaxed answer of a query. We propose an algorithm that incrementally generates matchings from a query to an RDF graph and also ranks tuples in the answer. In Section 4.1, we provide a procedure that computes the relaxation graph of a given triple pattern. In Section 4.2 we present an algorithm that efficiently computes the relaxed answer.

4.1 Computing Relaxations of a Triple Pattern

We first describe the computation of the relaxation graph for ontology relaxations. The procedure we propose is based on a variation of the notion of reduction of an RDFS ontology from [8]. The idea is to compute relaxed versions of a triple pattern t by applying the derivation rules (Figure 2) to t and and triples from the reduction. Given an ontology O, we denote by $\mathtt{red}(O)$ the RDF graph resulting as follows (reverse rule means deleting the triple deduced by the rule): (i) compute $\mathrm{cl}(O)$; (ii) apply reverse rules 2 and 4 until no longer applicable; and (iii) apply reverse rules 1 and 3 until no longer applicable. In what follows, we assume that $\mathtt{red}(O)$ has been precomputed.

We denote by $\Gamma(t)$ the set containing triples t' such that (i) there exists a triple $t_o \in \mathtt{red}(O)$ and $\frac{t, t_o}{t'}$ is an instance of a rule from groups (A), (B), or (C) (Figure 2), (ii) $\mathtt{var}(t') = \mathtt{var}(t)$, and (iii) $t' \notin \mathrm{cl}(O)$.

Proposition 3. *Let t be a triple pattern, such that $t \notin \mathrm{cl}(O)$ and $\mathtt{var}(t) \subseteq F$, where F is the set of fix variables. (i) $\{t' : t \prec_{\mathtt{onto}} t'\} \subseteq \Gamma(t)$. (ii) $\Gamma(t) \subseteq \{t' : t \prec^*_{\mathtt{onto}} t'\}$.*

Proposition 3 (i) does not longer hold if we reduce the ontology also using reverse rules 5 or 6. Proposition 3 (ii) follows directly from the definition of ontology relaxation. The set $\Gamma(t)$ can be easily computed in time $O(|\mathtt{red}(O)|)$ by searching for triples $t_o \in \mathtt{red}(O)$ such that t, t_o instantiates the antecedent of a rule, and testing the additional conditions given in the definition of Γ for the triple patterns derived. The relaxation graph of a triple t can be computed as follows. We start by computing $\Gamma(t)$, and in iteration i, we compute $\Gamma(t')$ with each new triple pattern t' obtained in iteration $i-1$, and add to the graph an edge (t', t'') for each $t'' \in \Gamma(t')$. In each iteration, we detect and delete transitive edges (an edge is transitive if it connects two nodes that are also connected by a path of length greater than one). In addition, we keep a list with triple patterns for which Γ has been already computed so that we do not repeat computations of Γ for the same triple pattern.

It is straightforward to generalize this procedure to compute direct simple relaxations. We just need to add in each iteration direct relaxations to triples that rename a constant with a variable or a variable (not in the original triple pattern t) with another variable. In each iteration, we also have to delete triple patterns that are isomorphic to some triple pattern already in the graph, and delete transitive edge.

Proposition 4. *Let $t = (a, p, b)$ be a triple pattern and O be an ontology. Let R be the relaxation graph of t. (i) R has $O(m^2)$ triples, where m is the number of triples in* $\texttt{red}(O)$. *(ii) Computing R takes time in $O(r^2 m)$, where r is the number of triples in R.*

From Proposition 4, it follows that the relaxation graph of a query has $O(m^{2n})$ nodes, where n is the number of triple patterns inside `RELAX` clauses in the query.

4.2 Computing the Relaxed Answer

In this section, we sketch a query processing algorithm which works by adapting the RDQL query processing scheme provided by Jena [15] to the processing of successive relaxations of a query. We assume the simplest storage scheme provided by Jena, in which the RDF triples are stored in a single table, called the *statement table.* The Jena query processing approach is to convert an RDF query into a pipeline of "find patterns" connected by join variables. Each triple pattern (find pattern in Jena's terminology) can be evaluated by a single SQL select query over the statement table. We formalize this with an operator called `find` that receives a triple pattern t and a statement table G and returns all matchings from t to the table.

In what follows, Q is the query whose relaxed answer we intend to compute, and Q' is an arbitrary query in the relaxation graph of Q. We have that $H = \texttt{Head}(Q) = \texttt{Head}(Q')$. For the sake of simplicity, we assume that each triple pattern in the body of Q is inside a `RELAX` clause. We assume that $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$, and $\texttt{Body}(Q') = \{t'_1, \ldots, \ldots, t'_n\}$. We also fix the statement table G we are querying. The answer of Q' can be computed by processing (in a pipelined fashion) a view, denoted $V_{Q'}$, defined by the following expression:

$$\pi_H(\texttt{find}(t'_1, G) \bowtie \ldots \bowtie \texttt{find}(t'_n, G)),$$

where π is the standard projection operator and $\bowtie$ is the natural join on variables shared by triple patterns. The answer of Q can be computed by a naive algorithm that traverses the relaxation graph of Q upwards, and in each step of the traversal, builds a view $V_{Q'}$, computes it, and returns those tuples which were not returned in previous steps.

Next, we propose an algorithm that avoids the redundant processing of tuples that arises with this naive approach. We define $\texttt{deltaFind}(t'_i, G)$ as the set containing triples $p \in G$ such that t'_i matches p, and no triple pattern directly below t'_i in the relaxation graph of t_i, matches p. The set $\texttt{deltaFind}(t'_i, G)$ can be computed similarly to $\texttt{find}(t'_i, G)$ by filtering triples from the statement table. Define a *delta view* for Q', denoted $\Delta_{Q'}$, as the following expression:

$$\pi_H(\texttt{deltaFind}(t'_1, G) \bowtie \ldots \bowtie \texttt{deltaFind}(t'_n, G)).$$

The following proposition shows that new answers (Section 3.5) correspond to delta views.

Proposition 5. *Let Q be a query and G be a RDF graph. For each query Q' in the relaxation graph of Q, (i) $\texttt{ans}(Q', G) = \bigcup_{Q_i : Q_i \prec^*_Q Q'} \Delta_{Q_i}(G)$, and (ii) $\texttt{newAnswer}(Q', G) = \Delta_{Q'}(G)$.*

The algorithm we propose (Figure 4), called `RelaxEval`, performs a breadth-first traversal of the relaxation graph of Q, building and processing each delta view $\Delta_{Q'}$ in each step of the traversal. The function *level* returns the level of a triple pattern t'_i in the relaxation graph R_i of t_i. Line 3(a) outputs the new answer of each query at level k. In order to find the queries at level k of the relaxation graph, the algorithm applies the following property. The queries Q' (defined by the join expression in Line 3 (a)) that belong to the level k of the relaxation graph of Q are those satisfying $\sum_i \mathit{level}(t'_i, R_i) = k$.

Algorithm `RelaxEval`

Input: a query Q (interpreted over an ontology O), where $\texttt{Body}(Q) = \{t_1, \ldots, t_n\}$, a statement table G, and an integer *maxLevel*.

Output: the set of tuples $\texttt{ans}_{\texttt{relax}}(Q, G, \mathit{maxLevel})$, where new answers are returned successively at each level of the relaxation graph.

```
1. $k := 0$, $\mathit{stillMore} := \mathit{true}$
2. For each triple pattern $t_i \in \texttt{Body}(Q)$, compute the relaxation graph $R_i$ of $t_i$ up to
   level $\mathit{maxLevel}$ (see Section 4.1).
3. While ($k \leq \mathit{maxLevel}$ and $\mathit{stillMore}$) do
   (a) For each combination $t'_1 \in R_1, \ldots, t'_n \in R_n$ such that $\sum_i \mathit{level}(t'_i, R_i) = k$
       do output $\pi_H(\texttt{deltaFind}(t'_1, G) \bowtie \ldots \bowtie \texttt{deltaFind}(t'_n, G))$
   (b) $k := k + 1$
   (c) $\mathit{stillMore}$ := exist nodes $t'_1 \in R_1, \ldots, t'_n \in R_n$ such that $\sum_i \mathit{level}(t'_i, R_i) = k$
```

Fig. 4. Algorithm that computes the relaxed answer of a query.

The algorithm `RelaxEval` induces a rank function, denoted $\texttt{rank}_{Q,G}$, which maps each tuple to the position at which `RelaxEval` returns it. The following proposition proves the correctness of `RelaxEval` (recall the notion of a consistent rank function from Section 3.5), where for a level k of the relaxation graph, we denote by $\texttt{RelaxEval}(Q, G, k)$ the set of tuples returned in Line 3(a) of `RelaxEval`.

Proposition 6. *Let Q be a query and G be a RDF graph. (i) For all k we have* $\texttt{RelaxEval}(Q, G, k) = \texttt{ans}_{\texttt{relax}}(Q, G, k)$. *(ii) The rank function* $\texttt{rank}_{Q,G}$ *is consistent.*

Both (i) and (ii) follow from Proposition 5 and the fact that the algorithm `RelaxEval` traverses the relaxation graph of Q in breadth-first fashion.

We end this section by comparing the computation cost of `RelaxEval` with the naive approach. We estimate the cost of computing a view $V_{Q'}$ as the expression $|\texttt{find}(t'_1, G)| \times \ldots \times |\texttt{find}(t'_n, G)|$, which represents the cost of the join operations. Roughly, it can be assumed that the `find` and `deltaFind` operations have the same cost, so we omit this cost in the expression. In the following proposition, we assume that $\delta = \frac{|\texttt{find}(t'_i,G)|}{|\texttt{deltaFind}(t'_i,G)|}$ is constant for every t'_i in the relaxation graph of every triple pattern $t_i \in \texttt{Body}(Q)$.

Proposition 7. *Let Q be a query (assume for simplicity that all its triple pattern are subject to relaxation), O be an ontology and G an RDF graph. (i) The naive approach to compute the relax answer at level k runs in time $O(\delta^n pT)$, where T denotes the time taken by* $\texttt{RelaxEval}(Q, G, k)$, *$n = |\texttt{Body}(Q)|$, and $p = |\texttt{relax}(Q, k)|$. (ii)* $\texttt{RelaxEval}(Q, G, k)$ *runs in time $O(m^{2n}|G|^n)$, where m is the number of triples in* $\texttt{red}(O)$.

The above proposition shows that the algorithm has exponential complexity, however its complexity is polynomial in the size of the data queried for a fixed query Q (data complexity). In addition, the answer is generated incrementally and hence the processing can be halted at any level in the relaxation graph. The number of triples in $\texttt{red}(O)$ provides an upper bound for k, the number of levels in the evaluation.

An improvement to the algorithm would be to process several delta views at the same time in an integrated pipelined fashion. In practice, we can improve query processing performance by further caching the results of $\texttt{deltaFind}(t, G)$ for all triple patterns t that occur more than once in the query relaxation graph (such duplicate occurrences can be detected as the relaxation graphs of the individual triple patterns in the original query are being constructed).

5 Related Work

Query languages based on regular expressions provide a form of flexible querying. The G$^+$ query language by Cruz et al. [6] proposes graph patterns where edges are annotated with regular expressions over labels. In this form, each graph pattern represents a set of more basic graph patterns, and therefore, a query extracts matchings that relate to its body in a variety of ways. This work considers queries over directed labeled graphs.

Kanza and Sagiv [11] propose a form of flexible querying based on a notion of homeomorphism between the query and the graph. Their data model is a simplified form of the Object Exchange Model (OEM).

Bernstein and Kiefer [1] incorporate similarity joins into the RDQL query language. This is done by allowing sets of variables in an RDQL query to be declared as *imprecise*. Bindings for these variables are then compared based on a specified similarity measure, such as edit distance.

Stuckenschmidt and van Harmelen [14] consider conjunctive queries over a terminological knowledge base that includes class, relation and object definitions. They also use query containment as a way of viewing query approximations, but are concerned about evaluating less complex queries first, so that the original query is evaluated last. They use a query graph to decide which conjuncts from the original query should be successively added to the approximate query. This is analogous to SPARQL queries in which every conjunct is optional.

Bulskov et al. [4] consider the language ONTOLOG which allows compound concepts to be formed from atomic concepts attributed with semantic relations. They define a similarity measure between concepts based on subsumption in a hierarchy of concepts. This gives rise to a fuzzy set of concepts similar to a given concept. They also introduce specialization/generalization operators into a query language that allow specializations or generalizations of concepts to be returned. They admit that combining this with similarity may make answers confusing.

6 Concluding Remarks

Despite being a relatively unexplored technique in the semantic Web, query relaxation may have an important role in improving RDF data access. One motivation for this technique is for querying data where there is a lack of understanding of the ontology that underlies the data. Another application is the extraction of objects with heterogeneous sets of properties because the data is incomplete or has irregular structure. As an example, a relaxed query can retrieve the properties that are applicable to each resource among a set of resources having different properties. Query relaxation can also make it possible to retrieve data that satisfies the query conditions with different degrees of exactitude.

There are several areas for future work. One is the introduction of relaxation into general SPARQL queries, including disjunctions and optionals. This should also involve a generalization of the `RELAX` clause so that it can be applied to entire graph patterns instead of single triple patterns. Another important issue for future work is the design, implementation and empirical evaluation of algorithms for computing relaxed answers. The graph-like nature of RDF provides additional richness for a query relaxation framework, which can be exploited in future work. For example, join dependencies between triple patterns of the query can be relaxed to connectivity relationships in RDF graphs.

References

1. A. Bernstein and C. Kiefer. Imprecise RDQL: Towards generic retrieval in ontologies using similarity joins. In *21th Annual ACM Symposium on Applied Computing (SAC/SIGAPP)*, Dijon, France, 2006.
2. D. Brickley and R. V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, 10 February 2004.
3. J. Broekstra. SeRQL: Sesame RDF query language. In *In M. Ehrig et al., editors, SWAP Deliverable 3.2 Method Design*, pages 55+68, `http://swap.semanticweb.org/public/Publications/swap-d3.2.pdf`, 2003.
4. H. Bulskov, R. Knappe, and T. Andreasen. On querying ontologies and databases. In *6th International Conference on Flexible Query Answering Systems*, pages 191–202, 2004.
5. K. G. Clark, editor. *RDF Data Access Use Cases and Requirements*, W3C Working Draft, 25 March 2005.
6. I. F. Cruz, A. O. Mendelzon, and P. T. Wood. A graphical query language supporting recursion. In *SIGMOD Conference*, pages 323–330, 1987.
7. T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *J. Intell. Inf. Syst.*, 1(3/4):293–321, 1992.
8. C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *23rd Symposium on Principles of Database Systems*, pages 95–106, 2004.
9. P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of RDF query languages. In *International Semantic Web Conference*, 2004.
10. P. Hayes, editor. *RDF Semantics*, W3C Recommendation, 10 February 2004.
11. Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *Symposium on Principles of Database Systems*, 2001.
12. F. Manola and E. Miller, editors. *RDF Primer*, W3C Recommendation, 10 February 2004.
13. E. Prud'hommeaux and A. Seaborne, editors. *SPARQL Query Language for RDF*, W3C Candidate Recommendation, 6 April 2006.
14. H. Stuckenschmidt and F. van Harmelen. Approximating terminological queries. In *5th International Conference on Flexible Query Answering Systems*, pages 329–343, 2002.
15. K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena. In *Proceedings of VLDB Workshop on Semantic Web and Databases*, 2003.