Combining Approximation and Relaxation in Semantic Web Path Queries

Alexandra Poulovassilis and Peter T. Wood

London Knowledge Lab, Birkbeck, University of London, UK
{ap,ptw}@dcs.bbk.ac.uk

Abstract. We develop query relaxation techniques for regular path queries and combine them with query approximation in order to support flexible querying of RDF data when the user lacks knowledge of its full structure or where the structure is irregular. In such circumstances, it is helpful if the querying system can perform both approximate matching and relaxation of the user's query and can rank the answers according to how closely they match the original query. Our framework incorporates both standard notions of approximation based on edit distance and RDFS-based inference rules. The query language we adopt comprises conjunctions of regular path queries, thus including extensions proposed for SPARQL to allow for querying paths using regular expressions. We provide an incremental query evaluation algorithm which runs in polynomial time and returns answers to the user in ranked order.

1 Introduction

The volume of semistructured data available to users on the web continues to grow, increasingly in the form of RDF linked data. Given the complexity and heterogeneity of such data, users may not be aware of its full structure and need to be assisted by querying systems which do not require that users' queries necessarily match exactly the data structures being queried.

In this paper we consider general semistructured data modelled as a graph, with RDF linked data being a particular application of this model. We are interested in developing efficient algorithms which allow for both *approximate matching* and *relaxation* of users' queries on such data, with the answers to queries being returned to users in ranked order. We restrict our query language to that of *conjunctive regular path queries* [2]. A conjunctive regular path (CRP) query Q consisting of n conjuncts is of the form

$$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_n, R_n, Y_n)$$

where each X_i and Y_i, $1 \leq i \leq n$, is a variable or constant, each Z_i, $1 \leq i \leq m$, is a variable appearing in the body of Q, and each R_i, $1 \leq i \leq n$, is a regular expression over the alphabet from which edge labels in the graph are drawn.

The answer to a CRP query Q on a graph G, $Q(G)$, is defined as follows. For each conjunct (X_i, R_i, Y_i), $1 \leq i \leq n$, let r_i be a binary relation over the scheme

(X_i, Y_i). Let $t[X_i]$ and $t[Y_i]$ denote the first and second components, respectively, of any tuple $t \in r_i$. There is a tuple $t \in r_i$ if and only if there exists a path from node $t[X_i]$ to node $t[Y_i]$ in G such that $t[X_i] = X_i$ if X_i is a constant, $t[Y_i] = Y_i$ if Y_i is a constant, and the concatenation of the edge labels in the path satisfies the regular expression R_i. Then $Q(G) = \pi_{Z_1,\ldots,Z_m}(r_1 \bowtie \cdots \bowtie r_n)$.

Using regular expressions to query data has been much studied, e.g. [2, 17], as have approximate query matching techniques, e.g. [4, 7, 14, 15, 18]. In [13], we studied a combination of these and showed that approximate matching of CRP queries can be undertaken in polynomial time. The edit operations we allowed in approximate matching of queries were insertions, deletions, substitutions, transpositions and inversions of edge labels (corresponding to reverse traversal of edges) — each with an assumed edit cost of 1. Here, for simplicity of exposition, we exclude inversions and transpositions — we note though that the techniques we develop here extend straightforwardly to this more general case, and the query complexity results still hold.

Example 1. The L4All system allows users to create and maintain a chronological record of their learning, work and personal episodes — their "timelines" — with the aim of supporting lifelong learners in exploring learning opportunities and in planning and reflecting on their learning [3]. Figure 1 illustrates a fragment of data and metadata relating to a user's timeline (where `sc` denotes `subclassOf`). The episodes within a timeline have a start and an end date associated with them (for simplicity these are not shown). Episodes are ordered by their start date — as indicated by edges labelled `next`. There are several types of episode, e.g. `University` and `Work`. Associated with each type of episode are several properties — we show just two of these, `qualif[ication]` and `job`.

Suppose that Mary is studying for a BA in English and wishes to find out what possible future career choices there are for her. Timelines may have edges labelled `prereq` between episodes, indicating that the timeline's owner believes that undertaking an earlier episode was necessary in order for them to be able to proceed to or achieve a later episode. So Mary might pose this query, Q_1[1]:

```
(?E2,?P)<-(?E1,type,University),(?E1,qualif.type,EnglishStudies),
          (?E1,prereq+,?E2), (?E2,type,Work), (?E2,job.type,?P)
```

However, this will return no results relating to the timeline of Figure 1, even though it is evident that this contains information that would be relevant to Mary. This is because, in practice, users may or may not create `prereq` metadata relating to their timelines. If Mary chooses to allow replacement of the edge label `prereq` in her query by the label `next`, she can submit a variant of Q_1:

```
(?E2,?P)<-(?E1,type,University), (?E1,qualif.type,EnglishStudies),
          APPROX(?E1,prereq+,?E2),(?E2,type,Work),(?E2,job.type,?P)
```

The regular expression `prereq+` can be approximated by the regular expression `next.prereq*` at edit distance 1 from `prereq+`. This allows the system to return

[1] In our assumed concrete syntax, variable names are preceded with '?'.

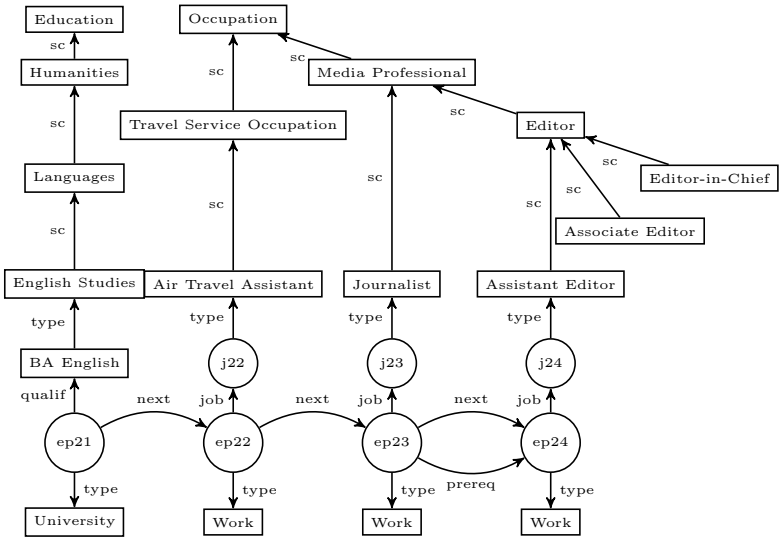

Fig. 1. A fragment of timeline data and metadata

the answer `(ep22,AirTravelAssistant)` at an edit distance 1 from Q_1. Mary may judge this not to be relevant and may seek further results, at a further level of approximation. The regular expression `next.prereq*` can be approximated by `next.next.prereq*`, now at edit distance 2 from Q_1, allowing the following answers `(ep23,Journalist)`, `(ep24,AssistantEditor)` to be returned. Mary may judge both of these as being relevant, and she can then request the system to return the whole of this user's timeline for her to explore further.

Suppose now Mary knows she wants to become an Assistant Editor and would like to find out how she might achieve this, given that she's done an English degree. Mary might pose this query, Q_2:

```
(?E2,?P)<-(?E1,type,University), (?E1,qualif.type,EnglishStudies),
          APPROX (?E1,prereq+,?E2), (?E2,job.type,?P)
          APPROX (?E2,prereq+,?Goal), (?Goal,type,Work),
          (?Goal,job.type,AssistantEditor)
```

At distance 0 and 1 there are no results from the timeline of Figure 1. At distance 2, the answers `(ep22,AirTravelAssistant)`, `(ep23,Journalist)` are returned, the second of which gives Mary potentially useful information.

Suppose Mary wants to know what other jobs, similar to an Assistant Editor, might be open to her. There are many categories of jobs classified under `Media Professional` but none of these will be matched by query Q_2 above. What she would like to pose instead (borrowing the 'RELAX' syntax of [12]) is query Q_3:

```
(?E2,?P)<-(?E1,type,University),(?E1,qualif.type,EnglishStudies),
          APPROX (?E1,prereq+,?E2),(?E2,job.type,?P)
          APPROX (?E2,prereq+,?Goal), (?Goal,type,Work),
```

`RELAX (?Goal,job.type,AssistantEditor)`

which would relax `Assistant Editor` to its parent concept `Editor`, matching jobs such as `Assistant Editor`, `Associate Editor`, `Editor-in-Chief` etc., as well as in parallel approximating the two instances of `prereq+`. Query results would be returned in increasing overall distance (relaxation and approximation) from the original query.

As a further extension, suppose another user, Joe, wants to know what jobs similar to being an Assistant Editor might be open to someone who has studied English or a similar subject at university. Subject disciplines are classified, e.g. `English Studies` under `Languages` which in turn is classified under `Humanities`. So Joe may pose query Q_4 which is identical to Q_3 above but with `RELAX` in front of `(?E1,qualif.type,EnglishStudies)`. □

In Section 2, we first consider computing approximate and relaxed answers for regular path queries consisting of a single conjunct. We show that in both cases answers can be computed in polynomial time in the size of the query and the input graph, and returned to the user in ranked order. Section 3 generalises to the case of multi-conjunct queries and shows that computation can still be achieved in polynomial time as long as the queries are acyclic and have a fixed number of head variables. In a multi-conjunct query, approximation and relaxation are combined by allowing each conjunct to be qualified by either an `APPROX` or a `RELAX` operator, as shown in the above example. Section 4 discusses related work. Section 5 presents our conclusions and future work.

2 Single-Conjunct Regular Path Queries

In this paper we consider a semistructured data model comprising a directed graph $G = (V, E)$ and an ontology $K = (V_K, E_K)$. V contains nodes representing entity instances or entity classes. E represents relationships between the members of V. Each node in V is labelled with a distinct constant. Each edge in E is labelled with a symbol drawn from a finite alphabet $\Sigma \cup \{\texttt{type}\}$. V_K contains nodes representing entity classes or properties. Each node in V_K is labelled with a distinct constant. We call a node in V_K representing an entity class a 'class node' and a node representing a property a 'property node'. So $V \cap V_K$ contains the set of class nodes of V. Each edge in E_K is labelled with a symbol drawn from $\{\texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}\}$. We assume that $\Sigma \cap \{\texttt{type}, \texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}\} = \emptyset$. We also assume that the set of labels of edges in E, except for the label `type`, is contained in the set of labels of property nodes in V_K. We observe that this general graph model encompasses RDF data, except that it does not allow for the representation of RDF's 'blank' nodes (but these are discouraged for linked data [10]). It also comprises a fragment of the RDFS vocabulary: `rdf:type`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain`, `rdfs:range`, which we abbreviate by `type`, `sc`, `sp`, `dom`, `range`.

A *single-conjunct regular path query* Q over a graph G is of the form:

$$vars \leftarrow (X, R, Y) \tag{1}$$

where X and Y are constants or variables, R is a regular expression over $\Sigma \cup \{\texttt{type}\}$, and $vars$ is the subset of $\{X, Y\}$ that are variables.

A *regular expression* R over $\Sigma \cup \{\texttt{type}\}$ is defined as follows:

$$R := \epsilon \mid a \mid \texttt{type} \mid \_ \mid (R1 \cdot R2) \mid (R1|R2) \mid R^* \mid R^+$$

where ϵ is the empty string, a is any symbol in Σ, "_" denotes the disjunction of all constants in $\Sigma \cup \{\texttt{type}\}$, and the operators have their usual meaning.

A *path* p in $G = (V, E)$ from $x \in V$ to $y \in V$ is a sequence of the form $(v_1, l_1, v_2, l_2, v_3, \ldots, v_n, l_n, v_{n+1})$, where $n \geq 0$, $v_1 = x$, $v_{n+1} = y$ and for each v_i, l_i, v_{i+1}, $v_i \xrightarrow{l_i} v_{i+1} \in E$. A path p *conforms* to a regular expression R if $l_1 \cdots l_n \in L(R)$, the language denoted by R.

Given a single-conjunct regular path query Q and graph G, let θ be a matching from variables and constants of Q to nodes of G, that maps each constant to itself. A tuple $\theta(vars)$ *satisfies* Q on G if there is a path from $\theta(X)$ to $\theta(Y)$ which conforms to R. The *answer* of Q on G is the set of tuples which satisfy Q on G. The answer can be found in polynomial time in the size of Q and G (from Lemma 1 in [17]).

Below we first briefly review approximate matching of single-conjunct regular path queries, from [13]. We then discuss relaxation of such queries based on information from the ontology K. Section 3 discusses combined approximation and relaxation for multi-conjunct queries.

2.1 Approximate Matching of Single-Conjunct Queries

The *edit distance* from a path p to a path p' is the minimum cost of any sequence of edit operations which transforms the sequence of edge labels of p to the sequence of edge labels of p' (note that edge labels are treated as atomic values and it is sequences of such labels that are transformed using edit operations). The edit operations that we consider here are insertions, deletions and substitutions of edge labels, each with an assumed edit cost of α, for some α.

The *edit distance* of a path p to a regular expression R is the minimum edit distance from p to any path that conforms to R. Given a matching θ from variables and constants of a query Q to nodes in a graph G, where constants must be matched to themselves, we say that the tuple $\theta(vars)$ has *edit distance* $\mathit{edist}(\theta, Q)$ to Q, and we define this to be the minimum edit distance to R of any path p from $\theta(X)$ to $\theta(Y)$ in G. Note that if p conforms to R, then $\theta(vars)$ has edit distance zero to Q.

The *approximate answer* of Q on G is a list of pairs $(\theta(vars), \mathit{edist}(\theta, Q))$, ranked in order of non-decreasing edit distance. The *approximate top-k answer* of Q on G comprises the first k tuples in the approximate answer of Q on G.

We now describe how the approximate answer can be computed in time polynomial in the size of R and G. The process is similar to that described in [13], but differs in a number of respects which are described below:

(i) We construct a *weighted* NFA M_R of size $O(R)$ to recognise $L(R)$, using Thompson's construction (which makes use of ϵ-transitions). M_R has set of

states S, alphabet $\Sigma' = \Sigma \cup \{\texttt{type}\}$, transition relation δ, start state s_0, and final state s_f. Each transition is labelled with a label from Σ' and a weight, or cost, which is zero in M_R. If X (or, respectively, Y) in the query is a constant n, we annotate s_0 (s_f) with n; otherwise we annotate s_0 (s_f) with a wildcard symbol $*$ that matches any constant.

(ii) We now construct the *approximate automaton* A_R corresponding to M_R. A_R has the same set of states as M_R, with the following additional transitions:
- For each state $s \in S$ and label $a \in \Sigma$, there is a transition (s, a, α, s), where α is the cost of insertion.
- For each transition $(s, a, 0, t)$ in M_R where $a \in \Sigma$, there is a transition (s, ϵ, α, t), where α is the cost of deletion.
- For each transition $(s, a, 0, t)$ in M_R, where $a \in \Sigma$, and label $b \in \Sigma$ $(b \neq a)$, there is a transition (s, b, α, t), where α is the cost of substitution.

Thus A_R has $O(|R| \cdot |\Sigma'|)$ transitions.

(iii) We form the weighted *product automaton*, H, of A_R with the graph $G = (V, E)$, viewing each node in V as both an initial and a final state. The states of H are of the form (s, n), $s \in S$ and $n \in V$.

(iv) To evaluate query Q, if X is a node v of G, we perform a shortest path traversal of H starting from the vertex (s_0, v). Whenever we reach a vertex (s_f, m) in H we output m, provided m matches the annotation on s_f. The distance of (v, m) to Q is given by the total cost of the shortest path from (s_0, v) to (s_f, m). If X is a variable, we perform such a traversal of H starting from vertex (s_0, v) for every node v of G.

This construction differs from that of [13] where the NFA for approximate matching of regular expression R was constructed using a number of copies of the NFA for recognising R, each corresponding to matching at a difference distance. Hence, in that NFA, distance was represented implicitly by the "copy number" of states, rather than explicitly using a weight as above. The use of annotations on states also does not appear in [13].

Proposition 1. *Let $G = (V, E)$ be a graph and Q be a single-conjunct query using regular expression R over alphabet Σ. The approximate answer of Q on G can be found in time $O(|R|^2|V|(|\Sigma'||E| + |V| \log(|R||V|)))$.*

The proof follows by using Dijkstra's algorithm on the product automaton H, which can be shown to have $O(|R||V|)$ nodes and $O(|R||\Sigma'||E|)$ edges.

The above query evaluation can also be accomplished "on-demand" by incrementally constructing the edges of H as required, thus avoiding precomputation and materialisation of the entire graph H. This is performed by calling a function `Succ` with a node (s, n) of H. The function returns a set of transitions $\stackrel{a,d}{\rightarrow} (p, m)$, such that there is an edge in H from (s, n) to (p, m) with label a and cost d. We show `Succ` below, where the function `nextStates`(A_R, s, a) returns the set of states in A_R that can be reached from state s on reading input a, along with the cost of reaching each. Note that we need either to remove ϵ-transitions from A_R (using a standard algorithm that potentially squares the

Procedure `Succ`(s, n)

Input: state s of A_R and node n of G
Output: set of transitions which are successors of (s, n) in H
$W \leftarrow \emptyset$
for $(n, a, m) \in G$ *and* $(p, d) \in \texttt{nextStates}(A_R, s, a)$ **do**
  add $\stackrel{a,d}{\rightarrow} (p, m)$ to W
return W

size of A_R) or `nextStates` needs to repeatedly follow ϵ-transitions until it finds a non-ϵ-transition, while summing costs of transitions.

A set $\texttt{visited}_R$ is maintained, storing tuples of the form (v, n, s) representing the fact that node n of G was visited in state s having started the traversal from node v. Also maintained is a priority queue $\texttt{queue}_R$ containing quadruples of the form (v, n, s, d), ordered by increasing values of d, where d is the distance associated with visiting node n in state s having started from node v. We begin by enqueueing the initial quadruple $(v, v, s_0, 0)$, if X is some node v, or enqueueing a set of initial quadruples otherwise, one for each node v of G. We maintain a list $\texttt{answers}_R$ containing tuples of the form (v, n, d) where d is the smallest distance of this answer tuple to Q and ordered by non-decreasing value of d. This list is used to avoid returning again (v, n, d') for any $d' \geq d$.

We then call a procedure `getNext` to return the next query answer, in order of non-decreasing distance from Q. `getNext` repeatedly dequeues the first quadruple of $\texttt{queue}_R$, (v, n, s, d), adding (v, n, s) to $\texttt{visited}_R$, until $\texttt{queue}_R$ is empty. After dequeueing the quadruple (v, n, s, d), we enqueue $(v, m, s', d + d')$ for each transition $\stackrel{e,d'}{\rightarrow} (s', m)$ returned by $\texttt{Succ}(s, n)$ such that $(v, m, s') \notin \texttt{visited}_R$. If s is a final state, its annotation matches n, and the answer (v, n, d') has not been been generated before for some d', then the triple (v, n, d) is returned.

2.2 Ontology Relaxation of Single-Conjunct Regular Path Queries

In [12], we considered relaxation of conjunctive queries over RDF data, and the formalisation of relaxation using RDFS entailment with respect to an RDFS ontology K. We assumed that the predicates of triples in K are in the set $\{\texttt{type}, \texttt{dom}, \texttt{range}, \texttt{sp}, \texttt{sc}\}$ and we adopted an operational semantics for the notion of RDFS *entailment*, denoted by $\models$ and characterised by the six rules shown in Fig. 2 (see [8, 9] for details).

We assumed infinite sets I (IRIs) and L (RDF literals). The elements in $I \cup L$ are called RDF *terms*. A triple $(v_1, v_2, v_3) \in I \times I \times (I \cup L)$ 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* is a set of RDF triples.

For RDF graphs G_1 and G_2, we stated that $G_1 \models_{\texttt{rule}} G_2$ if G_2 can be derived from G_1 by iteratively applying the rules of Fig. 2. We used the notion of the *closure* of an RDF graph G [9], denoted $\text{cl}(G)$, which is the closure of G under

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) $\frac{(a, \texttt{range}, c)\ (X, a, Y)}{(Y, \texttt{type}, c)}$

Fig. 2. RDFS Inference Rules

the rules. By a result from [9], RDFS entailment (for the fragment of RDFS we consider) can be characterized as follows: $G_1 \models_{\texttt{RDFS}} G_2$ if and only if $G_2 \subseteq \text{cl}(G_1)$.

Given a set of variables V disjoint from the sets I 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* P is a set of triple patterns. We denote the variables mentioned in P by $\texttt{var}(P)$.

A conjunctive query as considered in [12] is a rule whose body is a graph pattern. We investigated two broad classes of relaxations for such queries in that paper: *ontology relaxation* and *simple relaxation*. Ontology relaxation encompasses relaxations that are entailed using information from the ontology and are captured by the rules of Fig. 2; we note that when applying these rules to triple patterns, rather than (ground) triples, a, b and c must be instantiated to RDF terms, while X and Y can be instantiated to either RDF terms or variables. Simple relaxation consists of relaxations that can be entailed without an ontology, e.g. dropping triple patterns, replacing constants with variables, and breaking join dependencies.

In this paper, we extend the application of ontology relaxation from graph patterns to regular path queries, leaving consideration of simple relaxation to future work. Before proceeding further we introduce some assumptions and terminology.

We consider the cost of applying rule 2 or 4 to be β, and the cost of applying rule 5 or 6 to be γ. (Because queries and data graphs cannot contain `sc` and `sp`, rules 1 and 3 are inapplicable as far as relaxation is concerned.) We assume that the subgraphs of K induced by edges labelled `sc` and `sp` are acyclic; this ensures that the transitive reduction (see below) of each of these subgraphs is unique. We also assume that all the edges labelled with symbols from $\Sigma \cup \{\texttt{type}\}$ that are entailed by $G \cup K$ are included in G.

For each edge $(a, \texttt{type}, c)$ in G, we also add to G the "reverse" edge $(c, \texttt{type}^-, a)$. We do this because, while we do not consider reverse traversal of graph edges in general in this paper (leaving this as an area of further work), we do allow the reverse traversal of `type` edges, which we accommodate by generating reverse edges in G labelled $\texttt{type}^-$. We need these edges in order to accomodate Rule 6 of Fig. 2 without changing the position of the variable Y in the relaxed triple. This is because (as we will see below) in our context of relaxing regular path queries, the relaxed triples are generally part of a sequence of relaxed triples.

Thus, we use the equivalent form of $(c, \texttt{type}^-, Y)$ for the relaxed triple inferred by Rule 6.

Finally, we assume that $K = \texttt{extRed}(K)$, where $\texttt{extRed}(K)$ is the *extended reduction* of K. Given ontology K, $\texttt{extRed}(K)$ can be computed as follows: (i) compute $\text{cl}(K)$; (ii) apply the rules of Fig. 3 in reverse until no longer applicable; and (iii) apply rules 1 and 3 of Fig. 2 in reverse until no longer applicable. (Applying a rule in reverse means deleting the triple deduced by the rule.) Using this extended reduction allows us to perform what were termed *direct* relaxations in [12] which correspond to the "smallest' relaxation steps. This is necessary if we are to return query answers to users incrementally in order of increasing cost, which we discuss in more detail shortly.

$$(e1)\ \frac{(b, \texttt{dom}, c)\ (a, \texttt{sp}, b)}{(a, \texttt{dom}, c)} \qquad (e2)\ \frac{(b, \texttt{range}, c)\ (a, \texttt{sp}, b)}{(a, \texttt{range}, c)}$$

$$(e3)\ \frac{(a, \texttt{dom}, b)\ (b, \texttt{sc}, c)}{(a, \texttt{dom}, c)} \qquad (e4)\ \frac{(a, \texttt{range}, b)\ (b, \texttt{sc}, c)}{(a, \texttt{range}, c)}$$

Fig. 3. Additional rules used to compute the extended reduction of an RDFS ontology.

Let t_1 and t_2 be triple patterns such that $t_1, t_2 \notin \text{cl}(G \cup K)$, and $\texttt{var}(t_2) = \texttt{var}(t_1)$. We say that t_1 *relaxes to* t_2 (or t_2 is a *relaxation* of t_1), denoted $t_1 \leq t_2$[2], if $(\{t_1\} \cup G \cup K) \models_{\texttt{rule}} t_2$. Let P_1 and P_2 be graph patterns such that for all $t_1 \in P_1$ and $t_2 \in P_2$, $t_1, t_2 \notin \text{cl}(G \cup K)$ and $\texttt{var}(P_2) = \texttt{var}(P_1)$. We say that P_1 *relaxes to* P_2 (or P_2 is a *relaxation* of P_1), denoted $P_1 \leq P_2$, if for all $t_1 \in P_1$ there is a $t_2 \in P_2$ such that $t_1 \leq t_2$ and for all $t_2 \in P_2$ there is a $t_1 \in P_1$ such that $t_1 \leq t_2$. We note that the relaxation relation is reflexive and transitive.

Example 2. If we did not use the extended reduction of an ontology K, we could have the triples $(a, \texttt{dom}, c)$, $(a, \texttt{dom}, c')$ and $(c, \texttt{sc}, c')$ in K. Given a conjunct (X, a, w), we could apply rule 5 in order to relax (X, a, w) to $(X, \texttt{type}, c)$ with cost γ and to $(X, \texttt{type}, c')$, also with cost γ. However, the cost of relaxing (X, a, w) to $(X, \texttt{type}, c')$ should really be $\gamma + \beta$, reflecting the cost of using rule 5 to relax (X, a, w) to $(X, \texttt{type}, c)$ followed by the cost of using rule 4 to relax $(X, \texttt{type}, c)$ to $(X, \texttt{type}, c')$. The extended reduction of K does not contain the triple $(a, \texttt{dom}, c')$ because of applying rule e3 in reverse; hence, although the rules of Fig. 3 are not sound for RDFS entailment, using $\texttt{extRed}(K)$ allows us finer control over computing the cost of various relaxations. □

Given a query Q with a single conjunct (X, R, Y), let $q = l_1 l_2 \cdots l_n$ be a string in $L(R)$. We define a *triple form* of (Q, q) as a set of triple patterns

$$\{(X, l_1, W_1), (W_1, l_2, W_2), \ldots, (W_{n-1}, l_n, Y)\}$$

where $W_1, \ldots, W_{n-1}$ are variables not appearing in Q. Thus, a triple form of (Q, q) is a graph pattern which can be relaxed to another graph pattern.

[2] For notational simplicity we assume that the parameters G and K are implicit.

Example 3. Let query Q contain the single conjunct $(X, R, 4)$, where X is a variable, 4 is a constant, and $R = (a \cdot b \cdot d)$. Assume that K contains the triples $(d, \mathtt{sp}, e)$, $(e, \mathtt{dom}, c)$ and $(c, \mathtt{sc}, c')$. There is only a single $q \in L(R)$, namely $q = abd$. Consider the following triple form T of (Q, q)

$$\{(X, a, W_1), (W_1, b, W_2), (W_2, d, 4)\}$$

and let P be the graph pattern

$$\{(X, a, W_1), (W_1, b, W_2), (W_2, \mathtt{type}, c')\}$$

Then T relaxes to P since $(W_2, d, 4) \leq (W_2, \mathtt{type}, c')$ by applying rules 2, 5 and 4. We also have that $(W_2, d, 4) \leq (W_2, e, 4)$ (by rule 2), $(W_2, e, 4) \leq (W_2, \mathtt{type}, c)$ (by rule 5) and $(W_2, \mathtt{type}, c) \leq (W_2, \mathtt{type}, c')$ (by rule 4).

Note that, because of our requirement that variables be preserved when performing relaxation, rules 4, 5 and 6 can only be applied to the first or last triple pattern of a triple form of a string. So if, for example, $(b, \mathtt{dom}, f) \in K$, the triple pattern (W_1, b, W_2) cannot be relaxed to $(W_1, \mathtt{type}, f)$ by rule 5. □

We now define the *relaxed semantics* of such queries as follows. Let p be the path $(v_1, l_1, v_2, l_2, v_3, \ldots, v_n, l_n, v_{n+1})$, $n \geq 1$, in G. We define a *triple form* of p as a set of triple patterns

$$\{(v_1, l_1, W_1), (W_1, l_2, W_2), \ldots, (W_{n-1}, l_n, v_{n+1})\}$$

where $W_1, \ldots, W_{n-1}$ are variables. If p is of length zero, then p is of the form (v, ϵ, v) and the only triple form of p is also (v, ϵ, v).

Given a query Q of the form (1) and a graph G, let θ be a matching from variables and constants of Q to nodes of G such that θ maps each constant to itself. We denote $(\theta(X), R, \theta(Y))$ by $\theta(Q)$. Path p in G *r-conforms* to $\theta(Q)$ if there is a $q \in L(R)$, a triple form T_q of $(\theta(Q), q)$ and a triple form T_p of p such that $T_q \leq T_p$. A tuple $\theta(vars)$ *r-satisfies* Q on G if there is a path in G that r-conforms to $\theta(Q)$.

Note that a path in G can r-conform to a query on the basis of a triple pattern t relaxing to a triple pattern t' such that the constants in t and t' differ (due to applications of rules 5 and 6, provided Y is a constant). Hence relaxation of a conjunct induces a mapping on constants which may not be the identity.

We now consider the cost of applying relaxations in order to be able to return answers ordered by increasing cost. For this we need the notion of direct relaxation. In [12] we defined the *direct relaxation relation*, denoted by $\prec$, as the reflexive, transitive reduction of $\leq$. The *direct relaxations* of a triple pattern t (i.e., triple patterns t' such that $t \prec t'$) are the result of the smallest steps of relaxation. We write $t, o \vdash t'$ if t' can be derived from t and $o \in \mathrm{cl}(G \cup K)$ by the application of a single rule from Fig. 2. We also write $t, o \vdash_i t'$ if rule i was the rule used in the derivation.

It is shown in [12] that a single application of each of the rules in Fig. 2 to a triple pattern t and a triple $o \in \mathtt{extRed}(K)$ (where applicable) yields precisely

the direct relaxations of t with respect to K. Given graph patterns P_1 and P_2, we say that P_1 directly relaxes to P_2, denoted $P_1 \prec P_2$, if $P_1 = \{t_1\} \cup P$ and $P_2 = \{t_2\} \cup P$, for some (possibly empty) graph pattern P, and $t_1 \prec t_2$; in other words, $t_1, o \vdash_i t_2$ for some triple $o \in \texttt{extRed}(K)$ and rule i. The *cost* of the direct relaxation is the cost of applying rule i. The cost of a sequence of direct relaxations is the sum of the costs of each relaxation in the sequence.

Given ontology $K = \texttt{extRed}(K)$, path p in G, matching θ, query Q as in (1), string $q \in L(R)$, triple form T_q for $(\theta(Q), q)$, triple form T_p for p such that $T_q \leq T_p$ (so p r-conforms to $\theta(Q)$), the *relaxation distance* from p to $(\theta(Q), q)$ is the minimum cost of any sequence of direct relaxations which yields T_p from T_q. The cost of the empty sequence of direct relaxations (so that T_q is already a triple form of p) is zero. The *relaxation distance* from p to $\theta(Q)$ is the minimum relaxation distance from p to $(\theta(Q), q)$ for any string $q \in L(R)$.

Given graph G, query Q and matching θ, the *relaxation distance* of $\theta(Q)$, denoted $rdist(\theta, Q)$, is the minimum relaxation distance to $\theta(Q)$ from any path p that r-conforms to $\theta(Q)$. The *relaxed answer* of Q on G is a list of pairs $(\theta(vars), rdist(\theta, Q))$, where $\theta(vars)$ is an r-satisfying tuple, ranked in order of non-decreasing relaxation distance. The *relaxed top-k answer* of Q on G comprises the first k tuples in the relaxed answer of Q on G.

Example 4. Consider the conjunct $Q =$ `(?Goal,job.type,AssistantEditor)` from query Q_3 in Example 1. Suppose the graph G contains the triples `(ep24,job,j24)`, `(j24,type,AssistantEditor)` shown in Fig. 1, and also the triples `(ep33,job,j33)`, `(j33,type,AssociateEditor)` from another timeline. Path `(ep24,job,j24,type,AssistantEditor)` r-conforms to $\theta(Q)$ when $\theta($`?Goal`$)$ = `ep24` with relaxation distance 0. Path `(ep33,job,j33,type,AssociateEditor)` r-conforms to $\theta(Q)$ when $\theta($`?Goal`$)$ = `ep33` with relaxation distance β. So tuples (`ep24`) and (`ep33`) both r-satisfy Q on G. □

2.3 Computing the Relaxed Answer

We now describe how the relaxed answer can be computed, starting from the weighted NFA M_R that recognises $L(R)$ which was described in Section 2.1.

In computing a relaxed answer, it is useful to be able to make (possibly partial) copies of states in an automaton. Given an automaton M with a set of states S and a state $s \in S$, a *clone* of s in M is a new state s' which is added to S such that s' is an initial or final state if s is, and s' has the same sets of incoming and outgoing transitions as s. An *incoming (outgoing) clone* of s is a new state s' such that s' is an initial or final state if s is, s' has the same set of incoming (outgoing) transitions as s, and has no outgoing (incoming) transitions.

Given a weighted automaton $M = (S, \Sigma', \delta, s_0, s_f)$ and ontology K such that $K = \texttt{extRed}(K)$, we construct as described below the *relaxed automaton* $M^K = (S', \Sigma', \tau, S_0, S_f)$ of M with respect to K. The set of states S' includes S as well as any new states defined below. S_0 and S_f are sets of initial and final states, respectively, with S_0 including s_0, S_f including s_f and both possibly including additional cloned states defined below. Each state in S_0 and S_f is

annotated either with a constant or with the wildcard symbol $*$. The transition relation τ includes δ as well as any transitions added to τ by the process defined below. The process continues until no further changes to τ and S' occur.

- (rule 2) For each transition $(s, a, d, t) \in \tau$ and $(a, \mathtt{sp}, b) \in K$, add the transition $(s, b, d + \beta, t)$ to τ.
- (rule 4 (i)) For each transition $(s, \mathtt{type}, d, t) \in \tau$, $t \in S_f$ and $(c, \mathtt{sc}, c') \in K$ such that t is annotated with c, (i) add an outgoing clone t' of t annotated with c' to S', and (ii) add the transition $(s, \mathtt{type}, d + \beta, t')$ to τ.
- (rule 4 (ii)) For each transition $(s, \mathtt{type}^-, d, t) \in \tau$, $s \in S_0$ and $(c, \mathtt{sc}, c') \in K$ such that s is annotated with c, (i) add an incoming clone s' of s annotated with c' to S', and (ii) add the transition $(s', \mathtt{type}^-, d + \beta, t)$ to τ.
- (rule 5) For each $(s, a, d, t) \in \tau$, $t \in S_f$ and $(a, \mathtt{dom}, c) \in K$ such that t is annotated with a constant, (i) add an outgoing clone t' of t annotated with c to S', and (ii) add the transition $(s, \mathtt{type}, d + \gamma, t')$ to τ.
- (rule 6) For each $(s, a, d, t) \in \tau$, $s \in S_0$ and $(a, \mathtt{range}, c) \in K$ such that s is annotated with a constant, (i) add an incoming clone s' of s annotated with c to S', and (ii) add the transition $(s', \mathtt{type}^-, d + \gamma, t)$ to τ.

Given a regular expression R and ontology $K = \mathtt{extRed}(K)$, we denote by M_R^K the automaton obtained by first constructing the automaton M_R for R and then constructing the relaxed automaton of M_R with respect to K.

Example 5. Consider again conjunct $(X, R, 4)$, where $R = (a \cdot b \cdot d)$, and ontology $K = \{(d, \mathtt{sp}, e), (e, \mathtt{dom}, c), (c, \mathtt{sc}, c')\}$ from Example 3. The relaxed automaton M_R^K initially comprises the states $\{s_0, s_1, s_2, s_f\}$ and the transitions labelled with cost zero between them, as shown in Fig. 4. Applying the transformation for rule 2 to the transition labelled $d, 0$ and the triple $(d, \mathtt{sp}, e) \in K$, adds the transition labelled e, β from s_2 to s_f. Applying rule 5 to this transition and the triple $(e, \mathtt{dom}, c) \in K$, adds the outgoing clone s'_f of s_f, annotated with c, as well as the transition labelled $\mathtt{type}, \beta + \gamma$ from s_2 to s'_f. Applying rule 4(i) to this transition and the triple $(c, \mathtt{sc}, c') \in K$, adds the outgoing clone s''_f of s'_f, annotated with c', as well as the transition labelled $\mathtt{type}, 2\beta + \gamma$ from s_2 to s''_f.

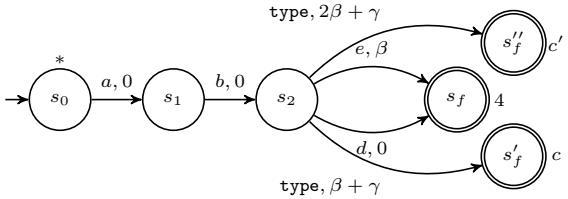

Fig. 4. Relaxed automaton M_R^K for conjunct $(X, (a \cdot b \cdot d), 4)$.

Given a graph G, automaton M_R^K will match (i) paths labelled $a \cdot b \cdot d$ from any node to node 4 with distance 0, (ii) paths labelled $a \cdot b \cdot e$ from any node to

node 4 with distance β, (iii) paths labelled $a \cdot b \cdot \texttt{type}$ from any node to node c with distance $\beta + \gamma$, and (iv) paths labelled $a \cdot b \cdot \texttt{type}$ from any node to node c' with distance $2\beta + \gamma$. □

Proposition 2. *Let Q be a query comprising a single conjunct (X, R, Y). Let $M_R^K = (S', \Sigma', \tau, S_0, S_f)$ be the relaxed automaton for regular expression R and ontology $K = \texttt{extRed}(K)$, where the ϵ-transitions have been removed from M_R^K. Let G be a graph and H be the product automaton of M_R^K and G. Let θ be a matching from Q to G such that $\theta(X) = v_0$ and $\theta(Y) = v_n$. (i) There is a path $r = (v_0, l_1, \ldots, l_n, v_n)$ in G that r-conforms to $\theta(Q)$ if and only if there is a path $p = ((s_0, v_0), (l_1, c_1), \ldots, (l_n, c_n), (s_n, v_n))$ in H, where $s_0 \in S_0$ and $s_n \in S_f$. (ii) Consider all paths of the form of p in (i). The relaxation distance from r to $(\theta(Q), q)$, where $q = l_1 \cdots l_n$, is given by the minimum value of $c_1 + \cdots + c_n$.*

The proof of (i) follows from the fact that the rules used to add transitions to M_R^K correspond to direct relaxations applied to triples. The proof of (ii) follows from the definition of relaxation distance.

Proposition 3. *Given a query Q comprising a single conjunct (X, R, Y) and ontology $K = \texttt{extRed}(K)$, the relaxed automaton M_R^K has at most $O(|R||K|)$ states and $O(|R||K|^2)$ transitions.*

The proof follows from the fact that automaton M_R contains $O(|R|)$ states, for each of which we can potentially add $O(|K|)$ cloned states. Each of the rules adds no more than $O(|K|)$ transitions for each of the $O(|R||K|)$ states in M_R^K.

Proposition 4. *Let $G = (V, E)$ be a graph, Q be a single-conjunct query using regular expression R, and $K = \texttt{extRed}(K)$ be an ontology. The relaxed answer of Q on G can be found in time $O(|R|^2|K|^2|V|(|E| + |V| \log(|R||K||V|)))$.*

The proof follows from Propositions 2 and 3, along with using Dijkstra's algorithm on the product automaton H, which can be shown to have $O(|R||K||V|)$ nodes and $O(|R||K|^2|E|)$ edges.

In order to compute the relaxed answers incrementally, we can use the `getNext` function from Section 2.1 along with the same initialisation of program variables. The only difference is that the `Succ` function now uses the relaxed automaton M_R^K rather than approximate automaton A_R.

3 General Queries

Combining the possibility of approximating and relaxing query conjuncts, a general query Q is of the form

$(Z_1, \ldots, Z_m) \leftarrow (X_1, R_1, Y_1), \ldots, (X_j, R_j, Y_j),$
$APPROX(X_{j+1}, R_{j+1}, Y_{j+1}), \ldots, APPROX(X_{j+k}, R_{j+k}, Y_{j+k}),$
$RELAX(X_{j+k+1}, R_{j+k+1}, Y_{j+k+1}), \ldots, RELAX(X_{j+k+n}, R_{j+k+n}, Y_{j+k+n})$

where $j, k, n \geq 0$, the X_i and Y_i are constants or variables, the R_i are regular

expressions, and each Z_i is one of $X_1, \ldots, X_{j+k+n}$ or $Y_1, \ldots, Y_{j+k+n}$. In the concrete syntax, conjuncts may be specified in any order.

Let θ be a matching from variables and constants of Q to nodes in graph G. The *distance* from θ to Q, $dist(\theta, Q)$, is defined as

$w_A(edist(\theta, (X_{j+1}, R_{j+1}, Y_{j+1})) + \cdots + edist(\theta, (X_{j+k}, R_{j+k}, Y_{j+k}))) +$
$w_R(rdist(\theta, (X_{j+k+1}, R_{j+k+1}, Y_{j+k+1})) + \cdots + rdist(\theta, (X_{j+k+n}, R_{j+k+n}, Y_{j+k+n})))$

where the coefficients w_A and w_R are set according to the preferences of the user. For example, they can be set to the same value if the same "cost" is associated with query approximation and query relaxation, or to different relative values to penalise one or the other more. Let $\theta(Z_1, \ldots, Z_m) = (a_1, \ldots, a_m)$. We call θ a *minimum-distance* matching if for all matchings ϕ from Q to G such that $\phi(Z_1, \ldots, Z_m) = (a_1, \ldots, a_m)$, $dist(\theta, Q) \leq dist(\phi, Q)$.

The *answer* of Q on G is the list of pairs $(\theta(Z_1, \ldots, Z_m), dist(\theta, Q))$, for some minimum-distance matching θ, ranked in order of non-decreasing distance. The *top-k answer* of Q on G comprises the first k tuples in the answer of Q on G.

The query Q can be evaluated by joining the answers arising from the evaluation of each of its conjuncts. For each APPROXed or RELAXed conjunct we can use the techniques described in Sections 2.1 and 2.3, respectively, to incrementally compute a relation r_i with scheme (X_i, Y_i, ED, RD). If $i \leq j$, then $t[ED] = t[RD] = 0$. If $j < i \leq j + k$, then for any tuple $t \in r_i$, $t[RD] = 0$ and $t[ED]$ is the edit distance for that tuple. If $j + k < i \leq j + k + n$, then for any tuple $t \in r_i$, $t[ED] = 0$ and $t[RD]$ is the relaxation distance for that tuple.

To ensure polynomial-time evaluation, we require that the conjuncts of Q are *acyclic* [6]. Hence a query evaluation tree can be constructed for Q, consisting of nodes denoting join operators and nodes representing conjuncts of Q. Given that the answers for single conjuncts are ordered by non-decreasing distance, we can use a pipelined execution of any rank-join operator, such as the recent instance-optimal FRPA operator proposed in [5], to produce the answers to Q on graph G in order of non-decreasing distance.

Example 6. Consider query Q_4 from Example 1. Suppose graph G contains the triples shown in Fig. 1 and also the triples
`(ep31,type,University),(ep31,qualif,BA History)`,
`(ep32,type,Work),(ep32,job,j32),(j32,type,Writer)`
`(ep33,type,Work),(ep33,job,j33),(j33,type,AssociateEditor)`
`(BA History,type,History),(ep31,next,ep32),(ep32,next,ep33)`
from another timeline. Suppose also that in the ontology, there are triples `(History,sc,Humanities)` and `(Writer,sc,MediaProfessional)`. We set the approximation cost $\alpha = 1$, the two relaxation costs $\beta = \gamma = 2$ and $w_A = w_R = 1$. Then, answers are produced for query Q_4 as shown in the table below:

?E1	?E1,RD	?E1,?E2,ED	?E2,?P	?E2,?Goal,ED	?Goal	?Goal,RD	?E2,P,D
ep21	*ep21,0*	ep23,ep24,0	*ep22,AT*	*ep23,ep24,0*	ep22	*e24,0*	*ep23,J,2*
ep31	**ep31,4**	*ep21,ep22,1*	*ep23,J*	ep21,ep22,1	*ep23*	**e33,2**	*ep22,AT,6*
		ep22,ep23,1	ep24,IE	*ep22,ep23,1*	*ep24*	*e23,4*	**ep32,W,8**
		ep31,ep32,1	**ep32,W**	ep31,ep32,1	ep32	e32,4	
		ep32,ep33,1	ep33,OE	**ep32,ep33,1**	**ep33**	e22,6	
		ep21,ep23,2		ep21,ep23,2			
		ep21,ep24,2		ep21,ep24,2			
		ep31,ep33,2		ep31,ep33,2			

The first seven columns refer to the answers produced for the individual conjuncts of Q_4. For brevity, we do not show the full four-attribute answer tuples, only the non-zero distances and the variable instantiations. We also abbreviate Air Travel Assistant by AT, Journalist by J, Writer by W, Assistant Editor by IE and Associate Editor by OE. The final column shows the overall query answers and distances. Tuples contributing to the first two answers are *italicised* and those contributing to the third answer are **bold**. □

4 Related Work

Various forms of query approximation and relaxation have been studied for a number of data models and query languages. For approximate querying, [14] considered querying semistructured data using flexible matchings which allow paths whose edge labels simply contain those appearing in the query to be matched. Such semantics can be captured by the edit operations of transposition and insertion. More generally, [7] used weighted regular transducers for performing transformations to regular path queries (but not CRP queries) to allow them to match semi-structured data approximately. The approximate queries of [18] are simply selections placed on attributes of form-based web data, where value constraints can be relaxed according to their perceived importance to the user.

In terms of query relaxation, work has been done on relaxing tree pattern queries for XML, recently in [16]. Relaxation of conjunctive queries on RDF is considered in [4, 12]. Rewriting rules are used on query patterns in [4] to perform both query refinement by including user preferences as well as query relaxation. Building on the work of [12], [11] develops a similarity measure for relaxed queries in an attempt to improve the relevance of answers. Similarity-based querying was also the focus of iSPARQL [15], where resources (rather than paths connecting them) are compared using similarity measures. Flexible querying of RDF using SPARQL and preferences expressed as fuzzy sets is investigated in [1].

In contrast to all the above, our work combines within one framework both query approximation and query relaxation, and applies it to the more general query language of conjunctive regular path queries on graph-structured data.

5 Concluding Remarks

We have discussed query relaxation for conjunctive regular path queries, and have shown how this can be combined with query approximation in order to provide greater flexibility in the querying of complex, irregular seminstructured data sets. Using the techniques proposed here, users are able to specify approximations and relaxations to be applied to their original query, and the relative costs of these. Query results are returned incrementally, ranked in order of increasing 'distance' from the user's original query. We have presented polynomial-time algorithms for incrementally computing the top-k answers to such queries.

In practice, we expect that a visual query interface would be required, providing users with readily understandable options from which to select their query

formulation, approximation and relaxation requirements, and set the relative cost associated with each operation they have selected. Our future work includes the design, prototyping and evaluation of such a query interface, or interfaces, and the empirical evaluation of our query processing algorithms, in domains such as querying of lifelong learners' metadata and heterogeneous medical data sets.

Another direction of ongoing research is to merge the APPROX and RELAX operations into one integrated 'FLEX' operation that applies concurrently both approximation and relaxation to a regular path query. For this, we are taking advantage of the common NFA-based approach that we have adopted.

References

1. P. Buche, J. Dibie-Barthélemy, and H. Chebil. Flexible SPARQL querying of web data tables driven by an ontology. In *Proc. FQAS*, pages 345–357, 2009.
2. D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. KR*, pages 176–185, 2000.
3. S. de Freitas, I. Harrison, G. Magoulas, A. Mee, F. Mohamad, M. Oliver, G. Papamarkos, and A. Poulovassilis. The development of a system for supporting the lifelong learner. *British Journal of Educational Technology*, 37(6):867–880, 2006.
4. P. Dolog, H. Stuckenschmidt, H. Wache, and J. Diederich. Relaxing RDF queries based on user and domain preferences. *J. Intell. Inf. Syst.*, 33(3):239–260, 2009.
5. J. Finger and N. Polyzotis. Robust and efficient algorithms for rank join evaluation. In *Proc. ACM SIGMOD*, pages 415–428, 2009.
6. G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 43(3):431–498, May 2001.
7. G. Grahne and A. Thomo. Regular path queries under approximate semantics. *Ann. Math. Artif. Intell.*, 46(1-2):165–190, 2006.
8. C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *Proc. PODS*, pages 95–106, 2004.
9. P. Hayes, editor. *RDF Semantics*, W3C Recommendation, 10 February 2004.
10. T. Heath, M. Hausenblas, C. Bizer, and R. Cyganiak. How to publish linked data on the web (tutorial). In *Proc. ISWC*, 2008.
11. H. Huang, C. Liu, and X. Zhou. Computing relaxed answers on RDF databases. In *Proc. WISE*, pages 163–175, 2008.
12. C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Query relaxation in RDF. *Journal on Data Semantics*, X:31–61, 2008.
13. C. A. Hurtado, A. Poulovassilis, and P. T. Wood. Ranking approximate answers to semantic web queries. In *Proc. ESWC*, pages 263–277, 2009.
14. Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *Proc. PODS*, pages 40–51, 2001.
15. C. Kiefer, A. Bernstein, and M. Stocker. The fundamentals of iSPARQL: A virtual triple approach for similarity-based semantic web tasks. In *Proc. ISWC*, pages 295–309, 2007.
16. C. Liu, J. Li, J. X. Yu, and R. Zhou. Adaptive relaxation for querying heterogeneous XML data sources. *Information Systems*, 35(6):688–707, 2010.
17. A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Computing*, 24(6):1235–1258, December 1995.
18. X. Meng, Z. M. Ma, and L. Yan. Answering approximate queries over autonomous web databases. In *Proc. WWW*, pages 1021–1030, 2009.