SPARQLeR: Extended Sparql for Semantic Association Discovery*

Krys J. Kochut and Maciej Janik

Department of Computer Science, University of Georgia
415 Boyd Graduate Studies Research Center
Athens, GA 30602-7404
{kochut,janik}@cs.uga.edu

Abstract. Complex relationships, frequently referred to as semantic associations, are the essence of the Semantic Web. Query and retrieval of semantic associations has been an important task in many analytical and scientific activities, such as detecting money laundering and querying for metabolic pathways in biochemistry. We believe that support for semantic path queries should be an integral component of RDF query languages. In this paper, we present SPARQLeR, a novel extension of the SPARQL query language which adds the support for semantic path queries. The proposed extension fits seamlessly within the overall syntax and semantics of SPARQL and allows easy and natural formulation of queries involving a wide variety of regular path patterns in RDF graphs. SPARQLeR's path patterns can capture many low-level details of the queried associations. We also present an implementation of SPARQLeR and its initial performance results. Our implementation is built over BRAHMS, our own RDF storage system.

1 Introduction

The size of ontologies in the Semantic Web has grown significantly within the last few years. The vision of ontologies containing millions of entities interconnected by meaningful relationships presented in [22] has become reality. The current query languages for RDF bases, such as SPARQL [24], RQL [14] and RDQL [21], support defining graph patterns and expressing various restrictions on entities and relationships participating in the defined patterns. However, all of them lack the necessary constructs that directly support the discovery of semantic associations, which cannot be explicitly defined by fully specified structure of a graph pattern.

We believe that querying for semantic associations is an important feature missing in the current RDF query languages, most notably in SPARQL. This paper presents SPARQLeR, a novel extension of SPARQL that enables the discovery of semantic associations among entities in RDF knowledge bases.

Semantic association is an undirected path that connects two entities in the knowledge base using named relationships, which represent its meaning. Discovery of

---

semantic associations is the process of finding paths of possibly unknown length that connect the given entities and have a specific semantics. Therefore, the search for paths must focus on the semantics of both the entities and the properties on the path. Moreover, the order of the relationships in the path and their directionality is crucial in expressing the semantics of the associations. To fulfill these requirements SPARQLeR uses regular expressions over properties for specifying the required semantics of the queried paths. The paths are treated as RDF meta-resources represented as sequences. They can be used in other patters, specifying the required properties of the individual path elements. This approach gives the user a detailed control over each of the elements on the path, as well as its overall semantics.

The paper is organized as follows. In Sec. 2, we give a motivation for adding semantic association discovery in RDF query languages. In Sec. 3, we discuss semantic associations and different types of paths in RDF bases. Sec. 4 introduces the concept of a path in SPARQLeR, shows the syntax and semantics of the language, and describes its prototype implementation. In Sec. 5, we discuss the initial performance results of our implementation.

2   Motivation

An important discovery in medicine made by Dr. D.R. Swanson of a dependency between Magnesium and Migraine [25] is a clear example of finding meaningful semantic associations. He manually searched through papers in PubMed [17] to establish a sequence of facts, supported by co-occurrence of significant terms in papers, that Magnesium may alleviate Migraine. With the suitable biomedical knowledge base extracted from PubMed and stored in RDF, as proposed in [19], finding such associations can be accomplished with the use of regular path queries.

Many interesting examples of semantic associations can be found in biological sciences. Metabolic pathways, composed of sequences of chemical reactions occurring within a cell, involve a gradual modification of the initial substance into the final product with the desired chemical structure.

N-Glycan Biosynthesis pathway [12] is an example of a well known metabolic pathway (presented later in Fig. 5). It starts from *dolichol phosphate* and ends with the production of *glyco peptide G00009*. It contains 15 chemical reactions and, even though this pathway may not be regarded as very long among the biochemical pathways, it is considered long for a path in the area of the Semantic Web.

Locating and retrieving metabolic pathways is a difficult problem. Regular path queries can be used searching for metabolic pathways. Using such queries, scientists should be able to query for and retrieve ordered sequences of specific reactions that lead from a given substance to a desired final product.

Additional interesting applications of semantic association discovery include BioPatentMiner [16] and Insider Threat [1]. We believe that there is a clear need for an RDF query language capable of semantic association retrieval.

Introduced in this paper SPARQLeR offers a variety of constructs for easy formulation of regular path queries which are suitable for solving the above problems.

3 Background

Path queries have been a focus of formal studies as well as practical applications. The complexity of finding regular paths in graphs was investigated in [15] and [7]. The authors showed that in general case finding all simple paths matching a given regular expression is NP-Complete, whereas in special cases it can be tractable. The complexity of various types of path queries, such as linear, regular and context-free was also described in [27]. Another approach was proposed in [6]. Here, the authors focused on finding paths in labeled graphs. In this case, a regular language is defined beforehand and a special index is maintained for all edge inserts and deletes.

Some of the query languages created for semi-structured databases support defining regular path queries. Among the well known languages are: G [10] and G+ [9], and Graphlog [8]. The relationship between the chain programs with recursive predicates and regular path queries is described in [4]. For RDF data, partial support for path queries, but not regular paths, can be found in SeRQL [5], TRIPLE [23], and Versa [18]. Versa introduced the *traverse* keyword which allowed querying for variable-length paths using a set of specified transitive properties. In [2], the authors present only the initial work on PSPARQL, a language supporting regular expressions in SPARQL. However, the regular expressions were to be used in place of properties in triple patterns, which limited the ability of testing individual path elements. It also significantly altered the syntax of SPARQL.

3.1 Semantic Associations in RDF Description Bases

Paths in RDF description bases represent a variety of explicit and implied semantic relationships among the participating resources (entities). This is based on the assumption that entities are semantically related if there exists a path connecting them. In [3], Anayawu and Sheth proposed a ρ-path (and related concepts) as a way of expressing semantic associations between entities in RDF bases. A ρ-path has been defined as a directed path connecting two entities.

While directed paths naturally capture semantic associations between entities, we also believe that undirected paths also capture important semantic associations which should not be ignored. Therefore, we view semantic associations as implied by the presence of either directed, undirected paths, or undirected paths with specific directionality of the included properties. A good illustration of this observation is an RDF graph, shown in Fig. 1, describing a part of a well known Glycan biosynthesis pathway (we discuss it further later in this paper). The shown fragment includes 3 reactions, represented by the entities *R05972, R05973,* and *R06238* and 4 glycans (*G00002-G00005)* as their reactants and products. For clarity of presentation, other properties have not been included in the shown graph.

The glycan *G00002* is a *predecessor* of *G00005*. Clearly, they are semantically associated, even though there is no directed path connecting them. In fact, the whole pathway links the starting substance, *dolichol phosphate*, and the final product, *peptide G00009*, using a sequence of reactions similar to the ones above. Again, a directed path connecting *dolichol phosphate*, and *peptide G00009* does not exist, but the two molecules are semantically related by this important pathway.
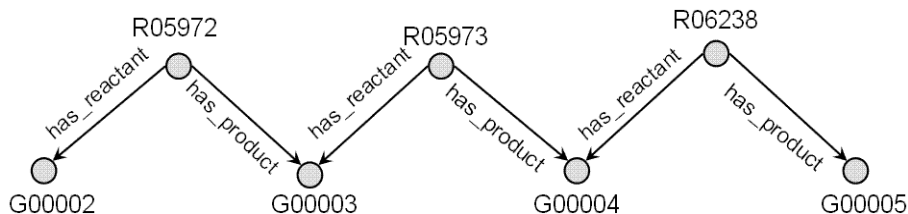
Fig. 1. An example of a chemical reaction graph

Below, we define semantic associations taking into account any type of connection between two entities. In what follows, we will interchangeably use the terms *triple* and *RDF statement*. We will assume that $\mathcal{R}$ is an RDF description base.

Def. 1. A *directed path* between resources r_0 and r_n in $\mathcal{R}$ is a sequence $r_0\ p_1\ r_1\ p_2\ r_2$, … $p_{n-1}\ r_{n-1}\ p_n\ r_n$ *(n>0)* if $r_0\ p_1\ r_1$, $r_1\ p_2\ r_2$, … $r_{n-2}\ p_{n-1}\ r_{n-1}$, $r_{n-1}\ p_n\ r_n$ *(n>0)* are triples in $\mathcal{R}$. The *length* of the path is *n*. Moreover, we require that all of the resources r_i ($0 \leq i \leq n$) in the path be distinct (we will only consider *simple paths*). □

Def. 2. An *undirected path* between the resources r_0 and r_n in $\mathcal{R}$ is a sequence $r_0\ p_1\ r_1$ $p_2\ r_2$, … $p_{n-1}\ r_{n-1}\ p_n\ r_n$ *(n>0)* if for each property and the two neighboring resources $r_{i-1}\ p_i\ r_i$ *(0 < i ≤ n)* in the path, either $r_{i-1}\ p_i\ r_i$. or $r_i\ p_i\ r_{i-1}$. is a triple in $\mathcal{R}$. We will consider only *simple undirected paths*. □

Def. 3. Two resources *r* and *s* in $\mathcal{R}$ are *semantically associated* if there exists an undirected path in $\mathcal{R}$ connecting the two resources. □

3.2 Defined Directionality Paths

While searching for semantic associations between two given entities we may be interested in paths in which properties follow a specific *defined directionality pattern*, according to the desired semantics of the connection between the entities. Creating such patterns requires an inverse property operator, not present in SPARQL. In SPARQLeR, we will use the '−' (minus) character to denote the inverse of a property.

Spatial relationships, such as *A is inside B*, offer illustrative examples for defined directionality paths. Let us consider the following three path queries with regular patterns (SPARQLeR's path patterns are defined later, in section 4.2):

1. *spatial:inside** - when used in a search for directed paths, it locates semantic associations illustrated by a diagram shown in Fig. 2a.
2. *spatial:inside** - when used in a search for undirected paths, it locates semantic associations illustrated by diagrams shown in Fig. 2a, 2b and 2c.
3. *(spatial:inside −spatial:inside)** [read as: concatenation of *inside* with *inverse of inside*] - when used in a search for directed paths, it locates semantic associations illustrated by a diagram shown in Fig. 2c, showing very specific, a chain-like inclusion structure.

Following the above observation, we believe that semantic associations require more than directed or undirected paths and should be treated as *defined directionality paths*.
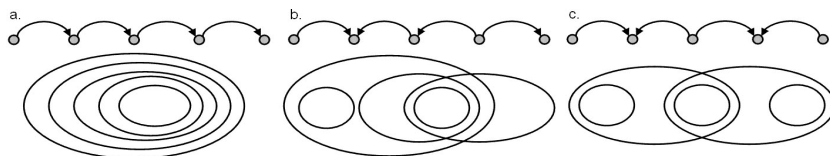
Fig. 2. Example results of spatial path queries

From a graph theoretical perspective, a path that matches a defined directionality pattern is an undirected path, and its implied semantics is set by the specific directionality of its member properties.

4   SPARQLeR

SPARQLeR (SPARQL *e*xtended with *R*egular paths) is an extension of SPARQL designed for querying for semantic associations. Our intension was to introduce minimal changes to SPARQL's syntax and semantics. Querying in SPARQLeR focuses on building *path patterns* involving undirected and directed paths as well as paths with defined directionality of the participating properties. Note, that since all properties have their inverses, the expressiveness of directed path queries is sufficient, as it enables us to construct undirected path patterns with the use of properties and their inverses. Nevertheless, to simplify the creation of path patterns, undirected path patterns are also supported. Syntax of proposed extensions fit seamlessly into current SPARQL language grammar. The new constructs in SPARQLeR are designed for the discovery of the semantic associations and, in particular, allow the user to:

- search for undirected paths or for paths with specific directionality of properties,
- filter located paths with the use of regular expressions formed over properties included in the path (use of inverse properties is also allowed),
- filter located paths by imposing constraints on the length of paths,
- filter located paths by requiring the presence of specific resources on the path, possibly even at a specific position,
- specify if located paths can include instance entities, schema classes and/or literals,
- indicate if the hierarchy of sub-properties should be used in property matching.

4.1   Path as RDF Meta-resource and Path Patterns

We will treat paths in RDF description bases as RDF *meta-resources*. In order to place these new meta-resources within the RDF vocabulary, we have created a new class *Path* defined in the new vocabulary *rdf-meta-schema*. The class Path has been defined as the sub-class of both rdf:Property and rdf:Seq as follows:

```
<rdf:Class rdf:about="http://meta.org/rdf-meta-schema#Path">
  <rdfs:isDefinedBy rdf:resource="http://meta.org/rdf-meta-schema#"/>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:subClassOf
    rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq"/>
  <rdfs:label>Path</rdfs:label>
  <rdfs:comment>The class of RDFMS paths.</rdfs:comment>
</rdf:Class>
```

According to the above definition, a resource of type *Path* is a sequence of other RDF resources. The sequence is composed of an ordered list of properties and connecting resources, as defined in Def. 2, but without the starting and ending resources, i.e. a path begins and ends with a property.

Having a path represented as a meta-resource of type rdfs:Seq, allows us to create patterns for inspecting the path elements with the use of the properties rdf:Member, and rdf:_*N*, which will be illustrated later in this paper.

A semantic association path between two resources *r* and *s* in an RDF graph forms a natural extension of a regular RDF property connecting the two resources. As such, the two resources and the path can be regarded as a *meta-triple* of the form *r path s*, where *path* is a meta-resource of type *Path*.

4.2   Path Patterns

We have extended SPARQL's triple patterns to include path patterns. Paths can be matched only by path variables. A *path variable* is a name beginning with the % character, for example *%connection.* A *path pattern* is a triple pattern created with the use of a path variable in place of the property. A *path query* is any SPARQL query involving at least one path pattern. Instead of formally defining SPARQLeR's syntax, we will present a number of examples illustrating path patterns.

The following SELECT path query, involving a *two-source path pattern*

```
SELECT %path WHERE {<r> %path <s>}
```

matches any path between the resources r and s. By default, the matched paths must be directed. As expected, for every matched path, the variable %path is bound to the located path, represented as a sequence (*rdf:Seq*) of properties and the connecting resources (the first and last elements of the sequence are properties). Therefore, the above query returns a list of blank nodes representing the matched paths (sequences).

In order to list the resources on each matched path, the *list* operator (applicable only to the path variables) must be applied to the path variable, as shown below:

```
SELECT list(%path) WHERE {<r> %path <s>}
```

Path patterns allow for searching for any resources reachable from a given one. The following *single source* SELECT query locates resources reachable from resource r:

```
SELECT %path, ?res WHERE {<r> %path ?res}
```

For every match, the reached resource is bound by the variable ?res, and both the path leading to it and the resource are returned by the query. The analogous form of the above query relies on the inverse path pattern of the form *{?res %path <r>}*. This pattern matches all resources (and paths) from which the resource r is reachable.

Since a matched path is a meta-resource of type *Path*, and therefore also of type rdf:Seq, resources on the path may be examined with the use of patterns involving the container membership properties. For example, the following query:

```
SELECT %path WHERE
{<r> %path <s> .   %path rdfs:Member <e>}
```

matches any semantic path between the resources r and s, provided the path includes the resource e. Even though it involves a path variable, the second triple is not a path triple, since the path variable is not used in place of the property. Similarly, we can formulate queries examining resources at specific positions on the path. For example, the following query:

```
SELECT %path WHERE {<r> %path <s> . %path rdfs:_1 <p>}
```

matches any semantic path between the resources r and s, provided the path begins with the property p (a SPARQLeR path always begins with a property). Similarly, the property *rdfs:_2* can be used to examine the first connecting resource on the path. In addition, two meta-properties *rdfms:entityResource* and *rdfms:propertyResource* (not described here) access the connecting resources and relationships, respectively (*rdfms* is the namespace prefix of the meta-schema, discussed in Sec. 4.1).

Path patterns may be also used in *construct*, *describe* and *ask* queries. As expected, a path variable used in a CONSTRUCT query returns all triples forming the paths matched by the querie's graph pattern. For example, the query

```
CONSTRUCT {<r> %path <s>} WHERE {<r> %path <s>}
```

returns all triples forming all paths between the resources r and s. The *list* operator cannot be used within the CONSTRUCT expression.

It is interesting to note that CONSTRUCT queries can be used to extract interesting sub-graphs, satisfying certain specific semantic properties. Combination of multiple path queries with use of CONSTRUCT, possibly with common intersecting points, may lead to creating semantically highly informative sub-graphs [20].

The ASK query functionality for path patters is defined as testing for existence of at least one specified path. The DESCRIBE query returns the description of all resources included in the found paths. DESCRIBE and ASK queries have not been included yet in presented implementation.

4.3 Testing Paths

Testing of the located paths can be performed with the use of special expressions used within the FILTER clause. A path can be tested if it matches a given regular expressions, or if its length is within certain bounds.

The *regex* operator in SPARQLeR has been extended to specify regular path expression filters. Syntactically, it is identical to the usual *regex* operator, but the first argument must be a path variable. The second argument must be a path expression, while the optional third argument specifies the path matching flags:

> regex(pathvar, pathexpr, pathflags)

The path expressions can be formed with the use of property names, their inverses, classes of properties, and the usual collection of regular expression operators. They are intended to specify the semantics of the path between a pair of resources.

Def. 4. SPARQLeR's *path expressions* are defined recursively as follows (p, p_1, p_2,…, and p_n denote property names, while x and y denote path expressions). We also define paths between resources r and s which are matched by the defined path expressions.

- *p*   matches a path between *r* and *s* of length 1 if a triple *r p s* exists;
- *-p* (the inverse of *p*) matches a path between *r* and *s* of length 1 if a triple *s p r* exists;
- [$p_1 p_2 \ldots p_n$] (class of properties) matches a path between *r* and *s* of length 1 if a triple *r p_i s* exists for some *i (1≤i≤n)*;
- -[$p_1 p_2 \ldots p_n$] matches a path between *r* and *s* of length 1 if a triple *s p_i r* exists for some *i (1≤i≤n)*; inverse operator is not allowed for properties inside the set;
- [^$p_1 p_2 \ldots p_n$] matches a path between *r* and *s* of length 1 if a triple *r p s* exists and $p \neq p_i$ *(1≤i≤n)*;
- -[^$p_1 p_2 \ldots p_n$] matches a path between *r* and *s* of length 1 if a triple *s p r* exists and $p \neq p_i$ *(1≤i≤n)*; inverse operator is not allowed for properties inside the set;
- . (wildcard) matches a path between *r* and *s* of length 1 if either triple *r p s* or *s p r* exists for some property *p*;
- also supported: *x | y* (alternative);   *xy* (concatenation);   *x** (Kleene star); *x+* (one or more repetition);   *(x)* (match a path matched by *x*).   □

For example, the following query matches paths between resources r and s that use only property *foo:prop*:

```
SELECT list(%path) WHERE
{<r> %path <s>
 FILTER(regex(%path,"foo:prop+")}
```

In order to keep the size of the path expressions manageable, only the prefix-abbreviated names of properties are allowed. The type of the located path (directed or undirected) can be requested as part of the *regex* expression and is indicated in the (optional) path flags of the *regex* expression. For example, the query

```
SELECT list(%path) WHERE
{<r> %path <s>
 FILTER(regex(%path,"(foo:prop|foo:rel)+","u")}
```

allows the matched path to be undirected. When the path directionality is left unspecified, the path is assumed to be directed (the flag "d" is assumed). Also, the path expression may be omitted, as in *regex(%path,,"u")*. Here, each path bound to variable %path may be undirected and be composed of arbitrary properties. A regex with no path expression is equivalent to *regex(%path,".*","u")*. Note, that *regex(%path,".*")* matches only directed paths, even though the wildcard expression (.) matches both a property and its inverse.

The other path flags include *i, s, l*, and *h*. The flags *i, s*, and *l* specify that the path is restricted to resources which are instances (entities), schema classes, and literals, respectively. The last flag, *h* (hierarchy)*,* indicates that when matching properties,additionally their ancestor properties (following the *subPropertyOf* property) may be used. The path flags may be combined. For example,

```
regex(%path,".*foo:prop.*","uis")
```

specifies that the path must involve property *foo:prop*, may be undirected, and can only involve connecting resources which are instances or schema classes. The default path flags string is *"di"*.

The new *length* operator returns the length of the path and can be used as part of a FILTER expression. For example,

```
SELECT list(%path) WHERE
{<r> %path <s>
 FILTER(length(%path)<5)}
```

restricts the matched paths to be of length less than 5. Due to implementation optimization, the length of a path may be compared only to constant values. Path filtering expressions may be combined, and mixed with any other filter tests, involving other variables and resources. As discussed in Sec. 0, the located paths may not be required to be fully directed, but with a specified directionality of individual properties. This may be requested by a suitable path expression, as in the following select query:

```
SELECT list(%path) WHERE
{<r> %path <s>
 FILTER(length(%path)<=6 && length(%path)>=4 &&
        regex(%path,"(foo:prop -foo:rel)+")}
```

which requires that the matched paths be composed of sequences of pairs of properties: *foo:prop* followed by the inverse of the property *foo:rel*.

4.4   Prototype Implementation of SPARQLeR

Our implementation of SPARQLeR uses BRAHMS, our own RDF storage system [13]. The implementation relies on BRAHMS's low level API to iterate over triples, depending on whether the subject, property, object, or their combination has already been fixed (by bound variables or explicit resources). The graph pattern included in a SPARQL query is converted into a composition of such iterators, according to a created query plan.

The path iterator, necessary for path pattern matching, has been implemented as a hybrid of a bidirectional breadth-first search and a simulation of a deterministic finite automaton (DFA) created for a given path expression. During our previous experiments [13], a bidirectional breadth first search proved to be the most efficient method in practice for finding all simple paths up to certain hop limit. For each instance of the iterator created for a path pattern, two DFAs are constructed. The first one accepts the regular language defined by the original path expression, while the second one accepts the reversed language, which is also regular. The path search uses the steps from the bidirectional BFS to grow the frontiers of entities used to connect paths. Before an entity is placed on the frontier for the next expansion, a check is performed if the partial path leading to it is not rejected by the appropriate DFA. This guarantees that the partial results, which are not accepted by DFA, will not be further expanded. Making this check for each node before adding it to a frontier causes the frontiers to grow very slowly for some regular expressions. From

the practical point of view, it significantly increases the possibility of finding longer paths in an acceptable amount of time and of not exhausting the memory used by the search.
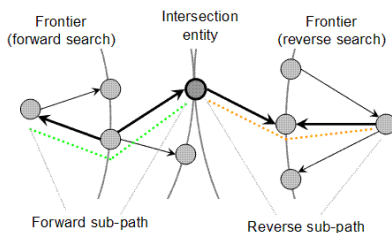

Fig. 3. Path finding and sub-paths in breadth-first search

A candidate path is located when an entity from the forward frontier matches an entity from the reverse frontier. At this point, it is only known that the "forward" sub-path has not been rejected by the forward DFA and that the "reverse" sub-path has not been rejected by the DFA accepting the reverse language. Before the concatenated path is returned, it must be accepted by the forward DFA, created from the original path expression.

A similar solution is used for single source path patterns. In this case, only one DFA in conjunction with a standard breadth first search is used to grow a single frontier of entities.

5   Experiment Design and Results

We have tested our implementation of SPARQLeR using a collection of path queries against a modified DBLP dataset [11]. We also performed path queries locating metabolic pathways in the Glycomics domain, using the GlycO ontology [26].

Tests were performed on machine with 2 Intel(R) Xeon(TM) 3.06GHz CPUs and 4Gb memory, running Red Hat 9.0 Enterprise Linux. C/C++ code was compiled using *gcc* (GCC) 3.2.3 20030502 (Red Hat Linux 3.2.3-56) with '–O6' optimization flag.

5.1   Data Sets

In our searching for metabolic pathways, we used the GlycO ontology. It represents information about glycans and includes a comprehensive schema as well as instances. GlycO is still under development and many new instances representing theoretical as well as experimental data are being added. Currently, the ontology has 362 classes (mainly glycans classification taxonomy) and 84 specialized properties.

Our scalability experiments required a much bigger data set. For this purpose we used a modified version of the DBLP ontology generated from the data available in September, 2006. It contains information about authors, published papers, articles, year of publication, etc. Unfortunately, the citations have been assigned to very few documents, rendering this set unsuitable for scalability test purposes. To be able to search for long, meaningful paths, we have replaced the current (few) citations with a

list of randomly created citations (1 to 10 random citations to papers selected from all of the previous years in the knowledge base, using a normal distribution). The total number of randomly inserted citations in the full dataset reached almost 4.3 million.

The full DBLP dataset contains 790,635 publications with set publish year. For scalability testing, we used a subset of publications published in or after 1981. It contains 760,369 publications and has been subdivided it into 26 subsets, each one including publications from an increasingly wider time range, starting with 2006 and ending with 1981 (the smallest set included only 2006 publications and the largest one included publications from years 1981-2006). The smallest test dataset contained almost 300,000 instance statements, while the largest one had over 6.6M instance statements. Fig. 4 presents numbers of publications in full DBLP (starting from year 1936) and sizes of used test datasets in statements.
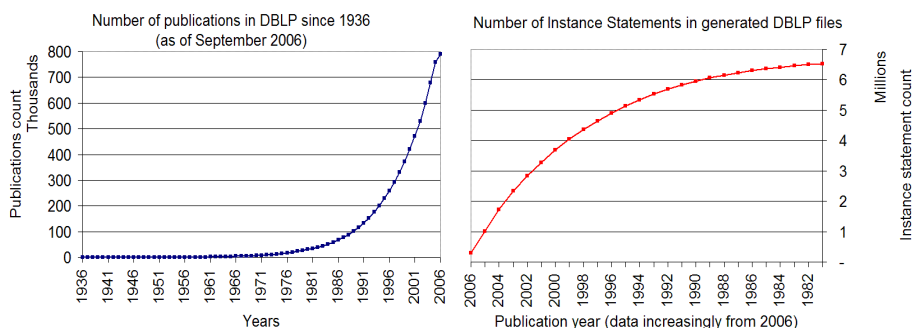

Fig. 4. Number of publications in DBLP from year 1936 and sizes of used test datasets

5.2 Functionality Test in the Biomedical Domain

We have tested the functionality of SPARLQeR on a wide selection of path queries, executed against a number of RDF bases. Due to the space limitations here, we will
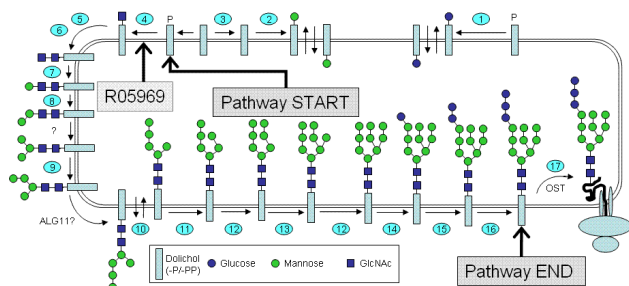

Fig. 5. N-Glycan biosynthesis pathway with query start and end points *(courtesy of Dr. Alison Vandersall-Nairn, University of Georgia)*

only discuss a particularly representative query in the biochemistry domain, retrieving a major part of the well known *N-Glycan biosynthesis pathway* [12]. The pathway is shown in Fig. 5 on the next page, where each arrow represents one reaction. The pathway is represented in GlycO, with the reactions represented as illustrated by the RDF graph in Fig 1.

We chose this pathway for its high regularity and a significant length. It enabled us to test if specifying paths using path expressions would help to find long, semantically relevant paths within an acceptable time. For this test we used the GlycO ontology and the SPARQLeR query used is presented below.

```
SELECT list(%path) WHERE {
  glyco:dolichol_phosphate %path glyco:glyco_peptide_G00009 .
  %path  rdfs:member  enzyo:R05969
  FILTER ( length(%path) <= 30 &&
    regex(%path, "((-glyco:has_acceptor_substrate|
      -glyco:has_reactant) glyco:has_product)*" ) ) }
```

This query located a pathway of length 30, consisting of 15 reactions. It starts with *dolichol phosphate*, goes through the reaction *R05969* (one of two possible at this step) and ends at *glyco peptide G00009*. Despite of the significant length, the result was retuned almost instantly, due to the high selectivity of path expressions. This proof of concept test demonstrated usefulness of the proposed SPARQL extension.

5.3 Scalability Tests on Modified DBLP Datasets

For the scalability tests, we randomly chose 14 papers published in 2006 and executed single-source queries to find all paths leading to papers they cited, using the relation *cites_publication*. A sample SPARQLeR query in presented below:

```
PREFIX opus: <http://lsdis.cs.uga.edu/projects/semdis/opus#>
SELECT ?end_publication WHERE {
  <http://dblp.uni-trier.de/rec/bibtex/journals/ai/Huber06>
  %path ?end_publication
  FILTER ( length(%path)<=26 &&
    regex(%path, "(opus:cites_publication)*" ) ) }
```

The queries were performed on increasingly larger datasets, starting with articles published only in 2006 and ending with articles published during 1981-2006. Each query was executed 4 times against each dataset. The plots in Fig. 6 on the next page present the execution time for all queries for each dataset and the number of located paths plotted on a logarithmic scale.

In the performed tests, the number of paths increased exponentially as the publications from the previous years were added. For the largest dataset, each query returned approximately 660,000 on average. The execution time also followed the exponential growth, but even for the longest query did not exceed 7 seconds.

Additionally, we performed tests for finding semantic associations between two given entities. We identified 4 early publications that were reachable by a relatively large number of paths from all previously chosen 14 starting publications. These 4
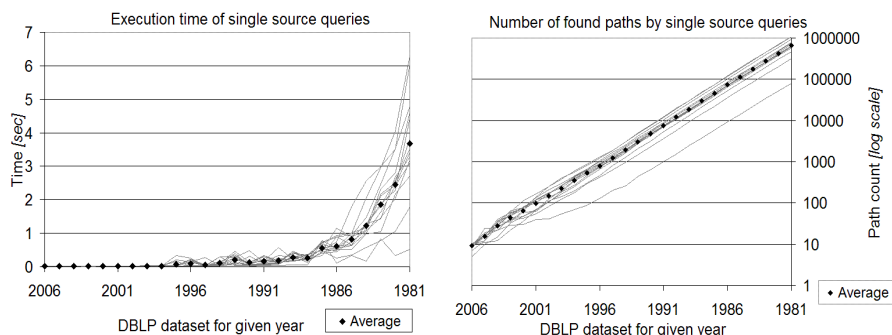
Fig. 6. Query execution times and number of found paths for single-source path queries

entities become endpoints for path queries between two resources. A sample SPARQLeR query is presented below:

```
SELECT list(%path) WHERE {
    <http://dblp.uni-trier.de/rec/bibtex/journals/ai/Huber06>
    %path
    <http://dblp.uni-trier.de/rec/bibtex/conf/programm/BarbutiM80>
    FILTER ( length(%path)<=26 &&
        regex(%path, "(opus:cites_publication)*" ) ) }
```

The queries were performed on increasingly larger datasets, while the length limit was increasing from 1 to 26, according to number of covered years in the datasets. For each of the 14 start entities we ran the path query to the 4 previously selected publications and averaged the results. The plots in Fig. 7 present the execution time and numbers of located paths for 14 start entities (each queried with 4 endpoints) for each dataset.
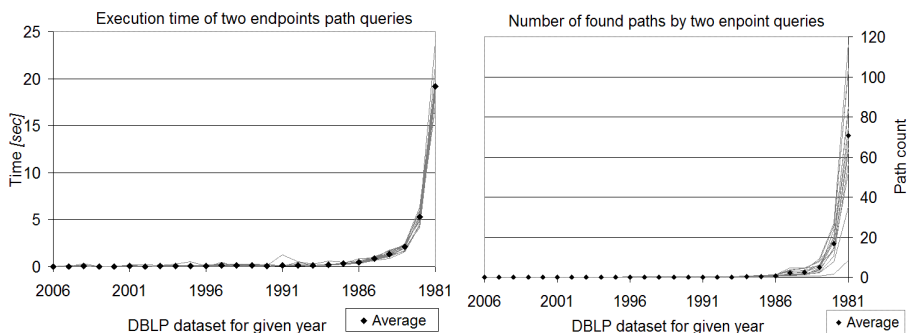

Fig. 7. Query execution times and number of found paths for path queries with set endpoints

In these tests, due to specificity of the dataset, although the number of results is a small fraction of previous ones, the search space became significantly larger than for the single-source queries. Nevertheless, the execution time did not exceed 25 seconds, which we think is a reasonable result for searching paths of length up to 26 hops. For shorter paths, the execution time drops drastically to below 1 second. In both cases,

such results for long queries can only mean that the given path expression was highly selective. It also proves the usability of the proposed SPARQL extension.

Of course, the path problem remains exponential and our solution does not change this fact. However, the results of our scalability experiments proved that in some practical cases, path queries can be solved within a reasonable amount of time, even for relatively long paths. This is possible with the use of path expressions which are highly selective with respect to a given dataset.

6   Conclusions and Future Work

We have presented SPARQLeR, a novel extension of SPARQL designed for finding semantic associations in RDF bases, and described its working implementation. SPARQLeR's path patterns have been seamlessly incorporated within SPARQL's graph patterns and allow for capturing both structural and semantic requirements of semantic association queries. Our experiments with path pattern queries have demonstrated the expressive power of SPARQLeR, effectiveness of its implementation, as well as its practical value in the presented examples.

Our future plans involve the optimization of regular path queries and incorporation of regular context into SPARQLeR. Despite the presented good timing results, we think that the optimization of path queries is very important for the practical use of the proposed language. This line of research involves not only optimization of simple queries, but of complex expressions and queries spanning multiple paths, as well.

We plan to base the notion of a context on our path patterns inducing RDF sub-graphs that will allow us to semantically specify a sub-graph of interest within an RDF description base. Consequently, this would support the idea that the same query executed in different contexts should return different results.

References

1. Aleman-Meza, B., Burns, P., Eavenson, M., Palaniswami, D. and Sheth, A., An Ontological Approach to the Document Access Problem of Insider Threat. in *IEEE International Conference on Intelligence and Security Informatics (ISI-2005)*, (Atlanta, Georgia, 2005).
2. Alkhateeb, F., Baget, J.-F. and Euzenat, J. Complex path queries for RDF *Poster paper in 4th International Semantic Web Converence (ISWC2005)*, Galway, Ireland, 2005.
3. Anyanwu, K. and Sheth, A., r-Queries: Enabling Querying for Semantic Associations on the Semantic Web. in *12th International World Wide Web Conf.*, (Budapest, Hungary, 2003).
4. Beeri, C., Kanellakis, P., Bancilhon, F. and Ramakrishnan, R., Bounds on the propagation of selection into logic programs. in *6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, (San Diego, California, United States, 1987), 214 - 226.
5. Broekstra, J. and Kampman, A. SeRQL: A Second Generation RDF Query Language *SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, Amsterdam, Netherlands, 2003.
6. Buchsbaum, A.L., Kanellakis, P.C. and Vitter, J.S., A data structure for arc insertion and regular path finding. in *1st annual ACM-SIAM symposium on Discrete algorithms*, (San Francisco, California, United States, 1990), 22-31.
7. Calvanese, D., Giacomo, G.D., Lenzerini, M. and Vardi, M.Y., Containment of Conjunctive Regular Path Queries with Inverse. in *7th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2000)*, (2000), 176-185.

8. Consens, M. and Mendelzon, A.O., Graphlog: a visual formalism for real life recursion. in *ACM Symposium On Principles of Database Systems*, (1990), 404-416.
9. Cruz, I.F., Mendelzon, A.O. and Wood, P.T., G+: Recursive queries without recursion. in *2nd International Conference on Expert Database Systems*, (1988), 355-368.
10. Cruz, I.F., Mendelzon, A.O. and Wood, P.T., A graphical query language supporting recursion. in *ACM SIGMOD International Conference on Management of Data*, (San Francisco, California, United States, 1987), ACM Press, 323-330.
11. Hassell, J., Aleman-Meza, B. and Arpinar, I.B. Ontology-Driven Automatic Entity Disambiguation in Unstructured Text *5th International Semantic Web Conference (ISWC-2006)*, Athens, GA, 2006.
12. Helenius, A. and Aebi, M. Roles of N-Linked Glycans in the Endoplasmic Reticulum. *Annual Review of Biochemistry*, *2004, 73*. 1019-1049.
13. Janik, M. and Kochut, K., BRAHMS: A WorkBench RDF Store And High Performance Memory System for Semantic Association Discovery. in *4th International Semantic Web Conference*, (Galway, Ireland, 2005).
14. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D. and Scholl, M., RQL: A Declarative Query Language for RDF. in *11th International World Wide Web Conference*, (Honolulu, Hawaii, USA, 2002), ACM.
15. Mendelzon, A.O. and Wood, P.T., Finding Regular Simple Paths In Graph Databases. in *15th Conference on Very Large Databases*, (Amsterdam, The Netherlands, 1989), Morgan Kaufman pubs. (Los Altos CA).
16. Mukherjea, S. and Bamba, B., BioPatentMiner: An Information Retrieval System for Biomedical Patents. in *13th International Conference on Very Large Data Bases (VLDB 2004)*, (Toronto, Canada, 2004), Morgan Kaufmann.
17. NLM. PubMed The National Library of Medicine, Bethesda MD.
18. Ogbuji, U. RDF Query using Versa Thinking XML: Basic XML and RDF techniques for knowledge management, Part 6, 10 April 2002.
19. Ramakrishnan, C., Kochut, K. and Sheth, A., A Framework for Schema-Driven Relationship Discovery from Unstructured text. in *5th International Semantic Web Conference (ISWC 2006)*, (Athens, Georgia, USA, 2006).
20. Ramakrishnan, C., Milnor, W.H., Perry, M. and Sheth, A.P. Discovering Informative Connection Subgraphs in Multi-relational Graphs. *SIGKDD Explorations*, *7* (2). 56-63.
21. Seaborne, A. RDQL - A Query Language for RDF, 2004.
22. Sheth, A., From Semantic Search & Integration to Analytics. in *Dagstuhl Seminar Proceedings 04391*, (Dagstuhl, Germany, 2005).
23. Sintek, M. and Decker, S. TRIPLE - An RDF Query, Inference, and Transformation Language *Deductive Databases and Knowledge Management*, Tokyo, Japan, 2001.
24. SPARQL. Query Language for RDF. Prud'hommeaux, E. and Seaborne, A. eds., 2005.
25. Swanson, R.D. Migraine and Magnesium: Eleven Neglected Connections. *Perspectives in Biology and Medicine*, *31* (4). 526-557.
26. Thomas, C.J., Sheth, A.P. and York, W.S., Modular Ontology Design Using Canonical Building Blocks in the Biochemistry Domain. in *International Conference on Formal Ontology in Information Systems (FOIS)*, (November 2006), IOS Press.
27. Yannakakis, M., Graph-theoretic methods in database theory. in *9th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, (Nashville, Tennessee, United States, 1990), ACM Press, 230-242.