The Fundamentals of iSPARQL: A Virtual Triple Approach For Similarity-Based Semantic Web Tasks

Christoph Kiefer, Abraham Bernstein, and Markus Stocker

Department of Informatics, University of Zurich, Switzerland
{kiefer,bernstein,stocker}@ifi.uzh.ch

Abstract. This research explores three SPARQL-based techniques to solve Semantic Web tasks that often require similarity measures, such as semantic data integration, ontology mapping, and Semantic Web service matchmaking. Our aim is to see how far it is possible to integrate customized similarity functions (CSF) into SPARQL to achieve good results for these tasks. Our first approach exploits virtual triples calling property functions to establish virtual relations among resources under comparison; the second approach uses extension functions to filter out resources that do not meet the requested similarity criteria; finally, our third technique applies new solution modifiers to post-process a SPARQL solution sequence. The semantics of the three approaches are formally elaborated and discussed. We close the paper with a demonstration of the usefulness of our iSPARQL framework in the context of a data integration and an ontology mapping experiment.

1 Introduction

Semantic Web tasks such as semantic data integration [18], ontology mapping [9], Semantic Web service matchmaking [14], and similarity-based retrieval [13] depend on some *notion of similarity.* Therefore, researchers still try to find sound *customized similarity functions* (CSF) to achieve good results for these tasks. Finding good similarity functions is, however, data-, context-, and sometimes even user-dependent, and needs to be reconsidered every time new data or a new task is inspected. Nonetheless, good CSFs are crucial for the success of the above-mentioned Semantic Web tasks.

In the past, we made the following two observations: First, it is often not enough to use a single similarity measure to achieve good results. In this case, a (possibly weighted) combination of atomic measures needs to be engineered (or even learned), which turns out to be best for the specific task and data [12]. In this paper, we, therefore, formally define the concept of a *similarity strategy* which can utilize a multitude of individual similarity measures and aggregation schemes to compare Semantic Web resources.

Second, in recent years, the RDF query language SPARQL has gained increasing popularity. It offers well-known constructs from database technology, defines standardized results formats, and is a protocol for distributed querying as well. The current W3C candidate recommendation of SPARQL [22] does, however, not support CSFs to analyze the data during query processing. This paper focuses on how to overcome this limitation by introducing iSPARQL – an extension of SPARQL that supports CSFs in order to query RDF graphs for similarities. The "i" stands for *imprecise* indicating that two or more resources are compared by using similarity measures. The proposed iSPARQL framework should be easy to use, easily extendable with new user-defined, task-specific similarity functions, as well as have a high degree of flexibility in terms of customization to the actual Semantic Web task.

In the following sections, we propose three approaches for integrating CSFs into SPARQL. Our first approach uses *virtual triples* calling property functions in the subject-predicate-object-style. These triples are not matched against the underlying ontology graph, but instead, employ CSFs to establish virtual relations (of similarity) between the resources under comparison. In that context, we will define the concept of *similarity joins* between RDF graph patterns which apply similarity measures to combine data. The second approach is based on pure SPARQL *extension functions* to filter out resources which are not sufficiently similar to each other. Our third method introduces new *solution modifiers* to the current SPARQL grammar to post-process and transform the solution of the graph pattern matching part into a new one by means of similarity measures. We evaluate our prototype iSPARQL system using two sets of experiments: (1) a *data integration experiment* and (2) an *ontology mapping experiment.*

Our contribution is, hence, twofold: first, *we present and compare three novel approaches for integrating similarity querying and SPARQL* resulting in a proposition for iSPARQL; second, *we show the importance of this proposition using two real-world Semantic Web tasks.*

The paper is structured as follows: next, we briefly introduce the most important related work, before we explain the details of our formal setup and the similarity measures used in the paper (Section 3). Section 4 explains our three approaches to add the notion of similarity into SPARQL, which are evaluated in Section 5. We close the paper with a brief discussion, limitations, some insights into future work, and our conclusions.

2 Related Work

Similarity. The concept of similarity is a heavily researched subject in Computer Science, Psychology, Artificial Intelligence, and Linguistics literature. Typically, those studies focus on the similarity between vectors [1], strings [7], trees [24], or objects [26]. In our case, we are interested in the similarity between (complex) Semantic Web resources of ontologies (*i.e.*, classes and individuals). Apart from this, several other studies reason generally about the theory of similarity (dissimilarity) [2, 20]. Most notably, Orozco and Belanche [20] define the concept of *similarity aggregation operators* – the counterpart to our *similarity aggregation schemes* – and its properties on a formal level.

SPARQL. Cyganiak [8] describes how to transform (a subset of) SPARQL into relational algebra, which is, as argued by the author, the language of choice when analyzing queries in terms of query planning and optimization. A translation into SQL is explained and the semantics of the relational algebra operators is defined. This idea is further refined by Pérez *et al.* [21] who conduct an extensive analysis of the semantics and complexity of SPARQL, focusing on the algebraic operators `JOIN`, `UNION`, `OPTIONAL`, and `FILTER`. The semantics and complexity of these operators are studied in great detail and insights into query optimization possibilities are presented. The foundations of SPARQL extension and property functions are, however, not addressed in their paper.

Siberski *et al.* [25] propose SPARQL extensions to allow the user to query RDF graphs with *user-defined preference criteria.* To achieve this goal, a new solution modifier is added to the official SPARQL grammar [22]. The main difference to our approach is the way the results are ranked: iSPARQL uses a multitude of similarity strategies (focusing on different dimensions of resources) to determine an overall degree of similarity between resources. To that end, it employs aggregation schemes to give more or less preference to one of the dimensions considered. Finally, the ranking is produced by ordering the results according to the overall similarity score. Thus, while preference criteria require an explicit formulation of one's preferences, which is oftentimes very difficult before seing the result set, iSPARQL allows their implicit determination via similarity measures – the approach preferred in Information Retrieval (IR).

In earlier work [4], we already introduced iRDQL– our extension of RDQL (SPARQL predecessor) with similarity joins. A limitation of iRDQL is that it allows the user to define only one similarity measure per query.

Similarity Joins (Data Integration). Our approach is partly inspired by studies from database research. Thus, we succinctly summarize the relevant publications. To perform data integration, Cohen [6] presents WHIRL and the notion of *similarity joins* by which data is joined on *similarity* rather than on *equality.* In WHIRL, the TF-IDF weighting scheme from IR [1] is applied together with the cosine similarity measure to determine the affinity of simple text in relations. Similar approaches are proposed by Gravano *et al.* [11] employing *text joins* to correlate information from different web sources.

In addition, a series of studies focuses on Semantic Web data integration: Noy [18] summarizes the necessity and requirements of ontology integration on the Semantic Web, pointing out the need for (semi-) automatic similarity detection between ontologies. This research resulted in the PROMPT Suite [19] to compare and align ontologies (among others). Furthermore, in two recent studies, Lam *et al.* [15] and Meštrović and Ćubrillo [17] propose their very specific approaches to Semantic Web data integration using the Flora-2 system and an RDF-enabled Oracle 10g database respectively.

Ontology Mapping. Euzenat *et al.* [10] propose an ontology alignment API and a Java tool call OLA that implements a universal measure for comparing resources in ontologies. Contributing to the same task, Ehrig *et al.* [9] present a layered approach to ontology mapping that focuses on different (modeling) aspects of ontologies. In their work, they define *similarity amalgamation functions* that are revived and studied in this paper, and denoted by *similarity aggregation schemes.*

3 Foundations: SPARQL and Similarity Measures

Before investigating different ways for adding the notion of similarity into SPARQL, we must lay out the foundations for this endeavor. Specifically, we need to discuss SPARQL and notions of similarity measures.

3.1 SPARQL and Similarity

In the heart of SPARQL, an RDF graph matching algorithm exhaustively tries to find mappings between query variables and graph nodes. According to [22], a solution mapping $\mu(?v \mapsto t)$ is defined as a mapping of a query variable $?v \in V$ to an RDF term t, where V is the infinite set of query variables and t a member of the set union of IRIs, RDF literals, and blank nodes. A multiset (or bag) of possible solution mappings is denoted by Ω. We define a similarity measure to be used in our iSPARQL approach (see Section 4) as follows:

Definition 1. *A similarity measure sm is a function* $sm : \mu_1 \times \mu_2 \mapsto \mathbb{R}$ *that associates the similarity of two input solution mappings* μ_1 *and* μ_2 *to a similarity score* $sc \in \mathbb{R}$ *in the range* $[0, 1]$.

In this context, a similarity score of 0 stands for complete inequality and 1 for equality of the input solution mappings μ_1 and μ_2.

Definition 2. *Solution mappings* $\mu_1(?v_1 \mapsto t_1)$ *and* $\mu_2(?v_2 \mapsto t_2)$ *are similar if the values* t_1 *and* t_2 *bound to their query variables are similar.*

In an iSPARQL query it should be possible to apply many different similarity measures, which would result in a set SC of individual similarity scores. When the similarity between (complex) Semantic Web resources should be calculated, it is desirable to combine different measures (*e.g.*, by using weights to give more or less importance to an individual similarity computation). For that purpose, we introduce our concept of a similarity aggregation scheme:

Definition 3. *A similarity aggregation scheme as is a function* $as : SC \mapsto \mathbb{R}$ *that defines how previously calculated similarity scores* $sc_i \in SC$*, where* $i \in \mathbb{N}$*, are combined. The result is again a similarity score* $sc_o \in \mathbb{R}$.

An aggregation scheme may combine the similarities using any type of mathematical function. It is left to the user who executes an iSPARQL query to consider the semantics of such functions. We can now define the concept of an iSPARQL similarity strategy:

Definition 4. *A similarity strategy st is a function* $st : SM \times AS \mapsto \mathbb{R}$ *that takes a set of similarity measures SM and aggregations schemes AS and returns a single similarity value sc expressing the combined, overall similarity between Semantic Web resources.*

Of course, an iSPARQL query can employ any number of different similarity strategies, which then, altogether, define an overall strategy to compare resources (refer to the extended query in Appendix A).

3.2 Similarity Measures

In all our experiments, we found it, a priori, hard to say which measure (or strategy) was best to be used in a query. Furthermore, the choice of the best performing similarity measure is often context- and data-dependent [3]. We, therefore, implemented a set of similarity measures that performed well in different application domains in a generic Java library called *SimPack*.[1] For the sake of completeness, we succinctly review the similarity measures we use in this paper: Levenshtein string similarity (*isparql:lev*), Jaccard similarity (*isparql:jac*), and TF-IDF (*isparql:tfidf*). Generally speaking, all measures of SimPack can be used in our iSPARQL framework.

The Levenshtein string similarity determines the relatedness of two strings in terms of the number of insert, remove, and replacement operations to transform one string str_1 into another string str_2 [16]. This edit distance is defined as $xform(str_1, str_2)$. As a normalization factor, the worst case transformation cost $xform_{wc}(str_1, str_2)$ is calculated replacing all parts of str_1 with parts of str_2, then deleting the remaining parts of str_1, and inserting additional parts of str_2. The final similarity between str_1 and str_2 is calculated by $sim_{lev}(str_1, str_2) = 1 - \frac{xform(str_1,str_2)}{xform_{wc}(str_1,str_2)}$ turning the normalized edit distance into a similarity score.

Co-occurrence measures are widely used for calculating similarity scores in text mining and IR [23]. The Jaccard measure calculates the similarity of two sets A and B as the ratio of the number of shared elements to the number of unified elements. For Jaccard, this is expressed by $sim_{jac}(A, B) = \frac{|A \cap B|}{|A \cup B|}$.

The TF-IDF measure aims at computing the degree of overlap of text documents as the cosine of the angle between the weighted vectors representing the documents [1]. TF-IDF gives each term t_i in a document d a weight which can be computed as $w_{t_i,d} = tf_{t_i,d} \times idf_{t_i} = tf_{t_i} \times log(\frac{N}{d_{t_i}})$, where $tf_{t_i,d}$ is the number of occurrences of t_i in d, N the total number of text documents, and d_{t_i} the number of documents where t_i appears. A high TF-IDF weight is reached by a high term frequency and a low inverse document frequency. Hence, common terms in the document collection are penalized. The similarity between two text documents is the cosine of the angle between their document vectors v_{d_1} and v_{d_2}: $sim_{tfidf}(\mathbf{v_{d_1}}, \mathbf{v_{d_2}}) = \frac{\mathbf{v_{d_1}} \cdot \mathbf{v_{d_2}}}{||\mathbf{v_{d_1}}||_2 \cdot ||\mathbf{v_{d_2}}||_2}$, where $||\mathbf{v}||_2$ is the L^2-vector norm.

4 Our Approach: Imprecise SPARQL

In this section, we present three approaches to the task of extending SPARQL with similarity operators: (1) the *virtual triple approach*: calling customized similarity functions (CSFs) that take some inputs and return an output to the query engine; (2) the *extension function approach*: using existing SPARQL filtering functionalities in combination with CSFs; and (3), the *solution modifier approach*: adding new solution modifiers to the official W3C SPARQL grammar to perform similarity computations.

The example query used in this section intends to find similar publications in two different datasets. It compares publication titles and book titles (*i.e.*, journal names) utilizing two similarity measures (*isparql:jac* and *isparql:lev*) and the *score*-aggregation scheme. This scheme sums up weighted similarity scores as follows: $\sum_{i=1}^{n} w_i sc_i$, where $i \in \mathbb{N}$ and $\sum_{i=1}^{n} w_i = 1$. The datasets are denoted by `opus` and `swrc` (see Section 5). The queries in the remainder of the paper use the prefixes shown in Listing 4.1.

[1] `http://www.ifi.uzh.ch/ddis/simpack.html`

```
PREFIX isparql: <java:isparql.>
PREFIX swrc:    <http://swrc.ontoware.org/ontology#>
PREFIX opus:    <http://lsdis.cs.uga.edu/projects/semdis/opus#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:     <http://www.w3.org/2002/07/owl#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
```

Listing 4.1. Query prefixes used in this paper.

4.1 Virtual Triple Approach

Our first proposed approach to solve the problem of adding similarity operators to SPARQL makes use of so called *magic properties*.[2] The concept behind this is simple: whenever the predicate of a triple pattern is prefixed with a special name, a call to a customized, external similarity function (CSF) is made and arguments are passed to the function (by the object of the triple pattern). Finally, a value is computed and returned to the subject variable of the triple pattern. We call this approach the *virtual triple approach* as such triple patterns are not matched against the underlying ontology graph, but against the only virtually existing similarity between the resources referred to in the triple. We define a virtual triple *vt* as a triple employing a particular kind of property function as follows:

Definition 5. *A virtual triple pattern vt is a triple of the form `[?v apf:funct ArgList]` where `funct` is a property function and `ArgList` a list of solution mapping arguments* $\mu(?x_1 \mapsto t_1), \mu(?x_2 \mapsto t_2), \ldots, \mu(?x_n \mapsto t_n)$ *of `funct`.*

A virtual triple establishes a relation between two Semantic Web resources, which is neither modeled in nor inferred using the typical RDFS or OWL semantics. The relation is entirely determined by the property function-defined logic and exists only during query execution (unless materialized in advance, see end of section). Virtual triples can conceptually be thought of as virtual relations such as `[?pub1 isSimilarTo ?pub2]` that would associate the two publication resources with a similarity score. Here, the predicate `isSimilarTo` does not have to exist in the RDF dataset D; it is, at this point, still considered as *imaginary*.

Syntax and Grammar. The query in Listing 4.2 shows the example query using virtual triples. The extended SPARQL grammar is shown in Table 4.1. To implement our virtual triple approach, we added a *SimilarityBlockPattern* symbol to the official SPARQL grammar rule of *GraphPatternNotTriples* [22]. The structure of *SimilarityBlockPattern* resembles the one of *OptionalGraphPattern* but has complete different semantics: instead of matching patterns in the RDF graph, the triples in an *SimilarityBlockPattern* act as virtual triple patterns, which are interpreted by the query processor. A *SimilarityBlockPattern* expands to rule [22.1] that adds the new keyword `IMPRECISE` to the grammar, which is followed by a number of virtual triples and optional `FILTER`-statements.

Semantics. The evaluation of the first four triple patterns (lines 3–6) results in four individual sets of solution mappings $\Omega_1, \ldots, \Omega_4$ that are successively tried to be joined. This operation is performed by extending the sets of mappings with compatible mappings from other sets (until all sets are processes and no more

[2] `http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions`

```
SELECT ?publication1 ?publication2 ?similarity
WHERE
  { ?publication1 rdfs:label ?title1 .
    ?publication1 opus:book_title ?booktitle1 .
    ?publication2 swrc:title ?title2 .
    ?publication2 swrc:booktitle ?booktitle2 .

    IMPRECISE {
        ?sim1 isparql:jac (?title1 ?title2) .
        FILTER (?sim1 >= 0.5) .
        ?sim2 isparql:lev (?booktitle1 ?booktitle2) .
        FILTER (?sim2 >= 0.5) .
        ?similarity isparql:score (?sim1 ?sim2 0.6 0.4) .
        FILTER (?similarity >= 0.5) }
  } ORDER BY DESC(?similarity)
```

Listing 4.2. iSPARQL example query for the virtual triple approach.

[22]	*GraphPatternNotTriples*	::= OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern | SimilarityBlockPattern
[22.1]	*SimilarityBlockPattern*	::= 'IMPRECISE' '{' ((VAR1 FunctionCall)+ Filter?)+ '}'

Table 4.1. Extended SPARQL grammar for the virtual triple approach.

compatible mappings are found). The semantics of this join operation of basic graph pattern matching is described in details by Pérez *et al.* in [21].

The semantics of a *SimilarityBlockPattern* is basically that of a *similarity join* and an (optional) *filter operation*: (1) it computes the query solutions for the similarity scores; (2) it eliminates those solutions which do not meet the filter constraints; and (3), it joins the remaining similarity scores to the solution mappings found by normal graph pattern matching.

We define the sets of *virtual solution mappings* μ_v as Ω_{VT} and the sets of solution mappings found by normal graph matching as Ω_{GPM}. Furthermore, on the basis of the definition of basic graph patterns in [22], we define *virtual graph patterns* VP as sets of virtual triple patterns *vt*. Based on this definitions, we define the similarity join between basic and virtual graph patterns as follows:

Definition 6. *A similarity join between basic and virtual graph pattern expressions P and VP extends the sets Ω_{GPM} returned from normal graph pattern matching with the sets of virtual solution mappings Ω_{VT} determined from virtual triple pattern matching.*

$$\Omega_{GPM} \ltimes_s \Omega_{VT} = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_{GPM},\ \mu_2 \in \Omega_{VT}\ AND\ \mu_2 \vDash R^-\}$$

Definition 6 accounts for *build-in* conditions R^-, which virtual solution mappings optionally must satisfy (denoted by $\mu_2 \vDash R^-$). In our case, possible conditions are constructed by using constants, virtual solution mappings, and the operators $<$, $\leqslant$, $\geqslant$, $>$, and $=$. If no conditions are defined, $\mu_2 \vDash R^-$ always evaluates to true. In accordance to Pérez *et al.* [21], we can now define the semantics of virtual graph pattern expressions using an evaluation function $[[\cdot]]$ over a dataset D.

$$[[vt]] = \{\mu_v(?v \mapsto sc) \mid sc = apf\text{:}funct\ (\ \mu(?x_1 \mapsto t_1), \ldots, \mu(?x_n \mapsto t_n)\ )\}$$

$$[[(P \text{ SIMJOIN } VP)]]_D = [[P]]_D \ltimes_s [[VP]] \qquad (1)$$

```
CONSTRUCT
  { ?publication1 simont:isSimilarTo ?publication2 . # similarity ontology
    ?publication1 simont:sc ?sim
  }
WHERE
  { ?publication1 rdfs:label ?title1 .
    ?publication2 swrc:title ?title2 .
    ?sim isparql:lev (?title1 ?title2) .
    FILTER ( ?sim >= 0.5 )
  }
```

Listing 4.3. CONSTRUCT-query to materialize similarity computations.

The first part of Equation 1 takes a virtual triple pattern expression and returns a set containing a single virtual solution mapping μ_v. In other words, a new solution mapping is generated that is not found by ordinary graph pattern matching and that assigns a similarity score to a query variable. Note that for a similarity measure, we limited `funct` to two input arguments, whereas more than two arguments can be passed to an aggregation scheme.

Pros and Cons. The following list summarizes the pros and cons of this approach.

- + Multiple similarity measures can be employed to compose sophisticated user- and data-specific similarity strategies.
- + Similarity scores are assigned to variables, thus, can be reused in the query for aggregation and ranking or can be returned for further processing.
- + Aggregation schemes can be applied to calculate overall similarity scores.
- − The SPARQL-grammar needs to be extended to account for the `IMPRECISE`-statements. This requires an adaptation of the query engines.
- − Queries using property functions depend on a query engine extension (currently only implemented in Jena ARQ[3]), hence, have limited interoperability.

Note that while we regard the need for extending SPARQL-engines with the iSPARQL grammar and property functions as the major downside of this approach, we think the benefits – mainly the possibility to establish virtual relations and to reuse similarity scores in the query for aggregation and ranking – are sufficient to justify such extensions.

Also, virtual triples can be materialized by using, for example, the SPARQL `CONSTRUCT`-query form in combination with a similarity ontology as sketched in Listing 4.3. This query constructs new triples as defined in the graph template of the query's construct-clause. It would require a similarity ontology (*simont*) that would specify all the classes and predicates necessary to model similarity calculations. Afterwards, the generated graph can be queried for the similarity scores with a common `SELECT`-query.

4.2 Extension Function Approach

We present a second approach that does not require to extend the SPARQL grammar. It is solely based on pre-defined SPARQL filter functions to carry out the desired similarity computations as part of the filtering process. Listing 4.4 shows the version of the example query that uses only extension functions to calculate and weight individual similarity scores.

[3] `http://jena.sourceforge.net/ARQ/`

```
SELECT ?publication1 ?publication2
WHERE
  { ?publication1 rdfs:label ?title1 .
    ?publication2 swrc:title ?title2 .
    ?publication1 opus:book_title ?booktitle1 .
    ?publication2 swrc:booktitle ?booktitle2 .

    FILTER ( isparql:jac(?title1, ?title2) => 0.5
          && isparql:lev(?booktitle1, ?booktitle2) >= 0.5
          && ( 0.6 * isparql:jac(?title1, ?title2)
             + 0.4 * isparql:lev(?booktitle1, ?booktitle2) >= 0.5 ) )
  }
```

Listing 4.4. iSPARQL example query for the extension function approach.

Semantics. The semantics is that of a SPARQL `FILTER`-expression with its filter condition R as defined in [21] but extended to account for extension functions. Thus, the set of operators to build the filter condition R has to be extended with the similarity measures and the symbols $<$, $\leqslant$, $\geqslant$, and $>$. We denote such a condition with similarity measures as R^+.

$$[[(P \text{ FILTER } R^+)]]_D = \{\mu \in [[P]]_D \mid \mu \vDash R^+\} \tag{2}$$

Pros and Cons. Assuming that there is a reference to the implementing class of the similarity measure, this approach can be used immediately with the current SPARQL specification. The similarity scores can, however, not be reused in the query. The list below summarizes the pros and cons of this approach.

+ No language extensions are necessary; all required features are already implemented in SPARQL.
+ Queries are interoperable with other SPARQL engines (assuming the engine can interpret the similarity measure specification referenced).
− Individual similarity scores cannot be assigned to variables. They, hence, cannot be reused in the query for aggregation and ranking.
− Aggregation schemes are more complex to compose as they have to be specified within filter expressions.
− The performance is likely to be suboptimal as similarity scores have to be calculated repeatedly (as long as no caching mechanisms are used).

4.3 Solution Modifier Approach

We mention a third approach more for completeness than to elaborate it in detail. The approach adds a new (complex) *solution modifier* to the official W3C SPARQL grammar [22]. Solution modifiers (aka *sequence solution modifiers*) (1) take the sequence of solutions (set of solution mappings) which is returned after the SPARQL graph pattern matching is finished, (2) modify it according to their semantics, and (3) return a new sequence of solutions to the user – essentially, they perform a post-processing step. This was the approach we suggested in iRDQL [4], except that iRDQL allows the user to define only one similarity measure per query. The query in Listing 4.5 shows our example using only solution modifiers to define similarity strategies.

We could not identify any benefits of this approach compared to the virtual triple or extension function approach mentioned previously. From a query design

```
SELECT ?publication1 ?publication2
WHERE
  { ?publication1 rdfs:label ?title1 .
    ?publication1 opus:book_title ?booktitle1 .
    ?publication2 swrc:title ?title2 .
    ?publication2 swrc:booktitle ?booktitle2
  }
IMPRECISE ( ?title1 ?title2 ) SIMMEASURE ( isparql:jac ) THOLD 0.5
IMPRECISE ( ?booktitle1 ?booktitle2 ) SIMMEASURE ( isparql:lev ) THOLD 0.5
AGGREGATOR ( isparql:score(0.6 0.4) ) THOLD 0.5
```

Listing 4.5. iSPARQL example query for the solution modifier approach.

point of view, data constraints should be specified in the WHERE-clause. Furthermore, solution modifiers can neither introduce new nor assign to existing query variables. Hence, there are no means to return the similarity scores to the user. Lastly, solution modifiers are not intended to access the ontology but only the result variables – an intention the similarity comparison between resources would break. Therefore, we decided to not further investigate this approach in this paper.

Summary In this section, we proposed three approaches for extending SPARQL with similarity joins. Given our elaborations about the pros and cons of all three approaches, we claim that the virtual triple approach is superior to the others as it (1) permits to return computed similarity scores (which neither of the other approaches does) and (2) allows the user to elegantly specify aggregations/combinations of such scores for customized similarity functions (which the extension function approach does not). Its major drawback is the use of virtual triples that some might deem as conceptually problematic. We disagree: in some sense, the specification of a similarity function is akin to the specification of an additional inferencing rule. *Hence, virtual triples can simply be regarded as part of the inferred knowledge base.*

5 Experiments: Illustrating the Power of iSPARQL

To show the power of the resulting iSPARQL framework, we performed two sets of experiments: (1) a *data integration experiment* – combining information from different RDF datasets, and (2) an *ontology mapping experiment* – aligning different ontologies along their class descriptions. In earlier works [12, 13], we already showed the power of iSPARQL for Semantic Web service matchmaking and similarity-based retrieval in large knowledge bases.

For both experiments, we used the SwetoDblp[4] and viewAIFB_OWL[5] datasets (with the prefixes opus and swrc). The former focuses on bibliography information of Computer Science publications and is based on DBLP,[6] whereas the latter is a collection of OWL annotations for persons, publications, and projects from SEAL (AIFB SEmantic portAL). For our ontology mapping experiments, we considered only the ontology schema files opus and swrc of SwetoDblp and viewAIFB_OWL respectively. Both files are available online at the respective project's websites.

[4] http://lsdis.cs.uga.edu/projects/semdis/swetodblp/

[5] http://www.aifb.uni-karlsruhe.de/viewAIFB_OWL.owl

[6] http://dblp.uni-trier.de/

opus:pub	**swrc:pub**	**opus:title**	**opus:bt**	**opus:date**	**swrc:pp**	**sim**
MikaISA04	id170instance	Ontology-based Content Management in a Virtual Organization	Handbook on Ontologies	2003-09-15	455-476	0.99
OberleS04	id206instance	The Knowledge Portal "OntoWeb"	Handbook on Ontologies	2004-03-30	499-517	0.94
OberleVSM04	id207instance	An Extensible Ontology Software Environment	Handbook on Ontologies	2004-03-30	311-333	0.93
SureSS04	id169instance	On-To-Knowledge Methodology (OTKM)	Handbook on Ontologies	2004-03-30	117-132	0.86

Table 5.1. Results of the data integration experiments.

5.1 Experiment 1: Semantic Web Data Integration

With the first set of experiments, we wanted to evaluate the applicability of our iSPARQL framework to the task of Semantic Web data integration. This task is highly relevant to distributed communities that want to integrate their heterogeneous knowledge bases (KB) to add, for example, cross-references, or to perform more sophisticated queries across different KBs to gain additional information. The problem arises when different parties have a different understanding of the same ontology for the same domain. This is a very typical situation in the Semantic Web, databases, and the Web in general: people working in the same or related field, model their domain ontologies (database schemas) similarly but, nevertheless, introduce syntactic, structural, and semantic differences.

In this section, we present our iSPARQL-based approach to Semantic Web data integration. To that end, we ran a slightly extended version of the query shown in Listing 4.2 on the two datasets SwetoDblp and viewAIFB_OWL. Note that this query is very similar to the one presented by Lam *et al.* [15] on Page 6 that aims at integrating publication information from different drug datasets. Whereas they have to rely solely on Oracle's SQL `regexp_like`-function,[7] iSPARQL can take advantage of a whole library of different similarity measures (*i.e.*, SimPack).

The query's final result set includes information from both datasets: title, proceedings name (bt), and a publication's last modification date from SwetoDblp, and the number of pages from viewAIFB_OWL, together with the similarity of the publications. The topmost results of the query based on similarity are shown in Table 5.1. All of the shown results provide the correct match between the two datasets. While this is not a statistical statement, we have shown the (statistical significant) usefulness of iSPARQL for the conceptually similar matchmaking and retrieval tasks elsewhere [5, 12, 13].

5.2 Experiment 2: Ontology Mapping

With our second set of experiments, we evaluated the applicability of our iSPARQL system to the task of ontology mapping. In other words, the task is to find classes in different ontologies, which *model the same real world concepts.*

To give a very simple example, we have chosen the query shown in Listing 5.1 that should find similar OWL classes in different ontologies. The query retrieves all resources which are of type `owl:Class`, filters out any anonymous nodes (complex classes), calculates the similarity of the class names using the Levenshtein

[7] Enabling regular expressions in queries.

```
SELECT ?OWLClass1 ?OWLClass2 ?similarity
FROM NAMED <http://lsdis.cs.uga.edu/projects/semdis/opus#>
FROM NAMED <http://swrc.ontoware.org/ontology#>
WHERE
  { GRAPH opus:
      { ?OWLClass1 rdf:type owl:Class .
        FILTER ( !isBlank(?OWLClass1) )
      }
    GRAPH swrc:
      { ?OWLClass2 rdf:type owl:Class .
        FILTER ( !isBlank(?OWLClass2) )
      }
    ?similarity isparql:lev ( ?OWLClass1 ?OWLClass2 ) .
    FILTER ( ?similarity > 0.65 )
  }
ORDER BY DESC(?similarity)
```

Listing 5.1. iSPARQL example query for the ontology mapping task.

OWLClass1	OWLClass2	similarity
opus:Proceedings	swrc:Proceedings	1.0
opus:University	swrc:University	1.0
opus:Masters_Thesis	swrc:MasterThesis	0.857
opus:Proceedings	swrc:InProceedings	0.846
opus:Book	swrc:InBook	0.667

Table 5.2. Results of the ontology mapping experiments.

similarity measure (see Section 3.2), and finally removes all solutions which are not sufficiently similar to each other.

The 5 topmost answers to the query are listed in Table 5.2 that shows classes of both ontologies together with their similarity scores. Note that we are aware of the extreme simplicity of this example. We think that it shows well, however, the potential of the iSPARQL framework. Imagine to use different similarity measures focusing on different modeling aspects of ontologies such as *data*, *structure*, and *context* [9], it is possible to compose much more complex similarity strategies than the one used in this experiment.

6 Limitations, Future Work, and Conclusions

In this paper, we have shown the syntactical and semantical foundations for a similarity join extension of SPARQL. A comparison of three such approaches, namely the virtual triple, the extension function, and the solution modifier approach has shown that the virtual triple approach is superior and provides SPARQL users with most flexibility in terms of defining customized similarity functions (CSF).

The major limitation of this approach lies in the need for extending existing SPARQL engines. As discussed above, we believe that the benefits warrants such an extension. Another limitation of the use of any similarity functions in SPARQL lies in the possibly enormous number of expensive (cross) joins involved during query execution. This problem can be addressed by (1) pre-computing similarity joins using a `CONSTRUCT`-statement as shown in Listing 4.3 or storing them in an index (as shown in [5]); or by (2) re-ordering the triples such that similarity joins will be executed only on subsets of the overall ontology as constrained by the query, which we also explored in [5].

Having implemented an iSPARQL query engine as an extension to Jena ARQ, we intend to further investigate the potential for iSPARQL query optimization beyond the first steps shown in [5]. Furthermore, we hope to embark on a systematic exploration of the suitability of similarity measures for different standard Semantic Web applications. Lastly, we intend to explore the possibility of extending description logic reasoners with customized similarity functions.

Whatever our further explorations will reveal, we firmly believe that the use of similarity functions is foundational for a large number of Semantic Web tasks and that this paper's discussion of the syntax and semantics of iSPARQL can provide a foundation for their use.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* 1999.
2. V. Batagelj and M. Bren. Comparing Resemblance Measures. *J. of Classification*, 12(1):73–90, 1995.
3. A. Bernstein, E. Kaufmann, C. Bürki, and M. Klein. How Similar Is It? Towards Personalized Similarity Measures in Ontologies. In *7. Int. Tagung WI*, pages 1347–1366, 2005.
4. A. Bernstein and C. Kiefer. Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In *Proc. of the ACM Symp. on Applied Computing*, pages 1684–1689, 2006.
5. A. Bernstein, C. Kiefer, and M. Stocker. OptARQ: A SPARQL Optimization Approach based on Triple Pattern Selectivity Estimation. Technical Report IFI-2007.02, Department of Informatics, University of Zurich, 2007.
6. W. W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 201–212, 1998.
7. W. W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of the Ws. on Information Integration on the Web*, pages 73–78, 2003.
8. R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, HP Labs, 2005.
9. M. Ehrig, P. Haase, N. Stojanovic, and M. Hefke. Similarity for Ontologies - A Comprehensive Framework. In *Proc. of the 13th Europ. Conf. on Information Systems*, 2005.
10. J. Euzenat, D. Loup, M. Touzani, and P. Valtchev. Ontology Alignment with OLA. In *Proc. of the 3rd Int. Ws. on Evaluation of Ontology-based Tools*, pages 59–68, 2004.
11. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text Joins in an RDBMS for Web Data Integration. In *Proc. of the 12th Int. World Wide Web Conf.*, pages 90–101, 2003.
12. C. Kiefer, A. Bernstein, H. J. Lee, M. Klein, and M. Stocker. Semantic Process Retrieval with iSPARQL. In *Proc. of the 4th Europ. Semantic Web Conf.*, pages 609–623, 2007.
13. C. Kiefer, A. Bernstein, and J. Tappolet. Analyzing Software with iSPARQL. In *Proc. of the 3rd Int. Ws. on Semantic Web Enabled Software Engineering*, 2007.
14. M. Klusch, B. Fries, and K. Sycara. Automated Semantic Web Service Discovery with OWLS-MX. In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 915–922, 2006.
15. H. Y. K. Lam, L. Marenco, T. Clark, Y. Gao, J. Kinoshita, G. Shepherd, P. Miller, E. Wu, G. Wong, N. Liu, C. Crasto, T. Morse, S. Stephens, and K.-H. Cheung. AlzPharm: integration of neurodegeneration data using RDF. *BMC Bioinformatics*, 8(3), 2007.

16. V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, 1966.
17. A. Meštrović and M. Ćubrillo. Semantic Web Data Integration Using F-Logic. In *Proc. of the 10th Int. Conf. on Intelligent Engineering Systems*, 2006.
18. N. F. Noy. What do we need for ontology integration on the Semantic Web, Position statement. In *Proc. of the 1st Semantic Integration Ws.*, pages 175–176, 2003.
19. N. F. Noy and M. A. Musen. The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. *Int. J. of Human-Computer Studies*, 59(6):983–1024, 2003.
20. J. Orozco and L. Belanche. On Aggregation Operators of Transitive Similarity and Dissimilarity Relations. In *Proc. of the IEEE Int. Conf. on Fuzzy Systems*, pages 1373–1377, 2004.
21. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. In *Proc. of the 5th Int. Semantic Web Conf.*, pages 30–43, 2006.
22. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report. W3C Candidate Recommendation 14 June 2007.
23. M. Rorvig. Images of Similarity: A Visual Exploration of Optimal Similarity Metrics and Scaling Properties of TREC Topic-Document Sets. *J. of the Am. Soc. for Inf. Sci.*, 50(8):639–651, 1999.
24. D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In *Pattern Matching in Strings, Trees, and Arrays*, pages 341–371. 1997.
25. W. Siberski, J. Z. Pan, and U. Thaden. Querying the Semantic Web with Preferences. In *Proc. of the 5th Int. Semantic Web Conf.*, 2006.
26. A. Tversky. Features of Similarity. *Psychological Review*, 84(2):327–353, 1977.

A Extended iSPARQL Example Query

```
SELECT ?publication1 ?publication2 ?similarity
WHERE
  { ?publication1 rdfs:label ?title1 .
    ?publication2 swrc:title ?title2 .
    ?publication1 opus:book_title ?booktitle1 .
    ?publication2 swrc:booktitle ?booktitle2 .

    IMPRECISE {
        ?sim1 isparql:jac (?title1 ?title2) .
        ?sim2 isparql:lev (?booktitle1 ?booktitle2) .
        ?sim3 isparql:score (?sim1 ?sim2 0.9 0.1) .
        FILTER (?sim3 > 0.2) }

    ?publication1 opus:hasAuthor ?author1 .
    ?publication2 swrc:author ?author2 .

    ?author1 foaf:lastname ?lastname1 .
    ?author2 foaf:lastname ?lastname2 .
    ?author1 foaf:firstname ?firstname1 .
    ?author2 foaf:firstname ?firstname2 .

    IMPRECISE {
        ?sim4 isparql:lev (?lastname1 ?lastname2) .
        FILTER (?sim4 >= 0.5) .
        ?sim5 isparql:lev (?firstname1 ?firstname2) .
        FILTER (?sim5 >= 0.5) .
        ?sim6 isparql:score (?sim4 ?sim5 0.7 0.3) }

    ?publication1 opus:abstract ?abstract1 .
    ?publication2 swrc:abstract ?abstract2 .

    IMPRECISE {
        ?sim7 isparql:tfidf (?abstract1 ?abstract2) .
        FILTER (?sim7 >= 0.7) .
        ?similarity isparql:average (?sim3 ?sim6 ?sim7) }
  }
```

Listing A.1. Extended iSPARQL example query for the virtual triple approach.