iRDQL - Imprecise RDQL Queries Using Similarity Joins

Abraham Bernstein
Department of Informatics
University of Zurich
Switzerland
bernstein@ifi.unizh.ch

Christoph Kiefer
Department of Informatics
University of Zurich
Switzerland
kiefer@ifi.unizh.ch

ABSTRACT

Traditional semantic web query languages support a logic-based access to the semantic web. They offer a retrieval (or reasoning) of data based on facts. On the traditional web and in databases, however, exact querying often provides an incomplete answer as queries are overspecified or the mix of multiple ontologies/modelling differences requires "interpretational flexibility." This paper introduces iRDQL – a semantic web query language with support for similarity joins. It is an extension to RDQL that enables the user to query for similar resources in an ontology. A similarity measure is used to determine the degree of similarity between two semantic web resources. Similar resources are ranked by their similarity and returned to the user. We show how iRDQL allows to extend the reach of a query by finding additional results. We quantitatively evaluated one measure of SimPack – our library of similarity measures for the use in ontologies – for its usefulness in iRDQL within the context of an OWL-S semantic web service retrieval test collection. Initial results of using iRDQL indicate that it is indeed useful for extending the reach of the query and that it is able to improve recall without overly sacrificing precision.

Categories and Subject Descriptors

H [**Information Systems**]: H.3 Information storage and retrieval; *H.3.3 Information Search and Retrieval*

General Terms

Similarity, RDQL, OWL-S, Retrieval, Evaluation

Keywords

Similarity Measures, Semantic Web, Similarity Joins, Imprecise Queries, Information Retrieval, Experimental Evaluation

K-CAP'05, October 2–5, 2005, Banff, Alberta, Canada.

1. INTRODUCTION

Imagine the following situation: You want to buy a used car – not just any car, but a car that has certain properties such as a minimum age, a favorite color, and a certain brand. All you have is a web interface that is connected to a large, semantically annotated database of cars, trucks, etc. When executing the query, however, you are buried in hundreds of results (or you may not get any answer if you overspecified the query). This situation is very typical. People querying the semantic web, databases, or also the web in general oftentimes find themselves either buried in results to their queries or with no results whatsoever. A common approach to handle these problems is to rank the results of a query, in the case of too many answers, or to return similar results, when no precise matches to the query exist [1, 4]. These two solutions require a measure of similarity between queries and answers. Finding a good measure of similarity is, thus, crucial for providing a good retrieval performance.

One means for querying ontologies is RDQL (RDF Data Query Language) [18, 19], which is a query language for RDF [15] in Jena models [5]. RDQL allows the user to formulate queries that return precise results. We extended RDQL with similarity joins [6] to retrieve not only the precise results of a query but also similar ones. In other words, our approach exploits the semantic annotation on the semantic web in conjunction with a similarity measure to improve the ranking of the results of queries for such resources. Thus, similar results may be found in the case where no precise results to a query exist. Additionally, if too many results are found, the intention of our approach is to use similarity measures to improve the ranking of the results.

We called our approach iRDQL. The i stands for *imprecise*, indicating that two or more resources do not have to be precisely equal but should be considered as equal with respect to their similarity value as computed by the similarity measure. The ranking of the results may be improved (or worsened) by specifying different similarity measures. Note that it is not necessarily clear which measure is best. On the contrary, the choice of the best

performing similarity measure is oftentimes context and data dependent [2, 8, 14]. We, therefore, implemented a set of similarity measures in a Java library that performed well in various application domains.

The contribution of this paper is our proposed iRDQL framework that extends traditional RDQL to allow the use of generic similarity elements. Such a generic element of an iRDQL query is for instance the similarity measure by which two resources are compared when the query is executed.

The paper is organized as follows: In Section 2 we give a brief introduction of SimPack [3], our library of implemented similarity measures for the use in ontologies. Section 3 explains our extensions to RDQL that make use of similarity joins and in Section 4 we illustrate the usefulness of our approach with the preliminary results from an evaluation, which uses a semantic web service retrieval test collection as data-set. We close the paper with some related and future work as well as conclusions.

2. SIMPACK

SimPack is a generic library of similarity measures implemented in Java. We found most of the similarity measures in the literature and adopted them for the use in ontologies. The library is generic, that is, the measures can be applied to different data sources and formats using data wrappers [7]. The question of similarity is a heavily researched subject in the computer science, artificial intelligence, psychology, and linguistics literature. Typically, those studies focus on the similarity between vectors [1, 16], strings [14], trees or graphs [20], or objects [8]. In our case we are interested in the similarity between resources in ontologies. Resources may be concepts (classes in OWL [13]) of some type or individuals (instances) of these concepts. The remainder of this section will discuss two types of measures.[1]

2.1 Vector-based Measures

One group of similarity measures operates on vectors of equal length. To simplify their discussion, we will discuss all measures as the similarity between the (binary) vectors $\mathbf{x}$ and $\mathbf{y}$, which are generated from the resources R_x and R_y of some ontology O. The procedure to generate these vectors depends on how one looks at the resources. If the resources are considered as sets of features (or properties in OWL), finding all the features for both resources results in two feature sets which are mapped to binary vectors and compared by one of the measures presented below in equations 1-5. For instance, if resource R_x has the properties *type* and *name* and resource R_y *type* and *age*, the following vectors $\mathbf{x}$ and $\mathbf{y}$ result using a trivial mapping M_1 from sets to vectors:

$$\mathbb{R}_x = \{type, name\} \Rightarrow \mathbf{x}' = \begin{pmatrix} 0 \\ name \\ type \end{pmatrix} \Rightarrow \mathbf{x} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\mathbb{R}_y = \{type, age\} \Rightarrow \mathbf{y}' = \begin{pmatrix} age \\ 0 \\ type \end{pmatrix} \Rightarrow \mathbf{y} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The typically used similarity measures for such vectors are the cosine measure, the extended Jaccard measure, and the overlap measure.

$$sim_{cosine}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2 \cdot ||\mathbf{y}||_2} \tag{1}$$

$$sim_{jaccard}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2^2 + ||\mathbf{y}||_2^2 - \mathbf{x} \cdot \mathbf{y}} \tag{2}$$

$$sim_{overlap}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\min(||\mathbf{x}||_2^2, ||\mathbf{y}||_2^2)} \tag{3}$$

In these equations, $||\mathbf{x}||$ denotes the L^1-norm of $\mathbf{x}$, i.e. $||\mathbf{x}|| = \sum_{i=1}^{n} |x_i|$, whereas $||\mathbf{x}||_2$ is the L^2-norm, thus $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$. The cosine measure quantifies the similarity between two vectors as the cosine of the angle between the two vectors whereas the extended Jaccard measure computes the ratio of the number of shared attributes to the number of common attributes [21]. In addition, the Dice measure [11] and the Euclidean distance are also implemented in SimPack.

$$sim_{dice}(\mathbf{x}, \mathbf{y}) = \frac{2 \cdot \mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||_2^2 + ||\mathbf{y}||_2^2} \tag{4}$$

$$d_{euclid}(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||_2 \tag{5}$$

The metric Euclidean distance is converted from a distance (dissimilarity) to a similarity measure using the formula [11]

$$sim_{dist}(\mathbf{x}, \mathbf{y}) = \frac{1}{1 + d_{dist}(\mathbf{x}, \mathbf{y})} \tag{6}$$

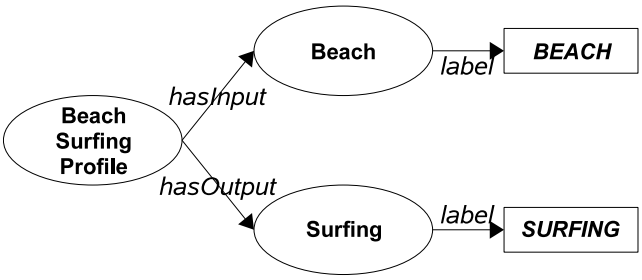
where $dist \in \{euclid, manhattan, \dots\}$ is one of the Minkowski distances ($p = 1$ for manhattan, $p = 2$ for euclid) given by

$$L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |\mathbf{x}_i - \mathbf{y}_i|^p \right)^{\frac{1}{p}} \tag{7}$$

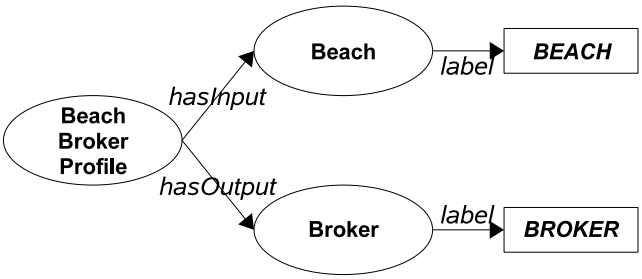
2.2 Sequence-based Measures

A different mapping M_2 from the feature set of a resource makes use of the underlying RDF-graph representation of ontologies. In this mapping, a resource R is considered as starting node to traverse the graph along its edges where edges are properties of R connecting other resources. These resources in turn may be concepts or, eventually, data values. To illustrate the mapping, refer to Figures 1.1 and 1.2 that show two different

[1] We have also introduced a formal framework of concepts and individuals in ontologies but omit it here due to space constraints. Please refer to [3] for further reading about the formal framework.



1.1: *Beach Surfing Profile*



1.2: *Beach Broker Profile*

Figure 1: Partial semantic network representations of two resources *Beach Surfing Profile* and *Beach Broker Profile*

individuals *Beach Surfing Profile* (R_x) and *Beach Broker Profile* (R_y). Recursively traversing these individuals, starting at nodes *Beach Surfing Profile* and *Beach Broker Profile* results in the following two feature sets:

$$\mathbb{R}_x = \{BeachSurfingProfile, hasInput, Beach, label, BEACH, hasOutput, Surfing, label, SURFING\}$$

$$\mathbb{R}_y = \{BeachBrokerProfile, hasInput, Beach, label, BEACH, hasOutput, Broker, label, BROKER\}$$

Here, these sets are considered as vectors of strings, $\mathbf{x}$ and $\mathbf{y}$ respectively. The similarity between strings is often described as the edit distance (also called the Levenshtein edit distance [10]), the number of changes necessary to turn one string into another string. Here a change is typically defined as either the insertion of a symbol, the removal of a symbol, or the replacement of one symbol with another. Obviously, this approach can be adapted to strings of concepts (i.e., vectors of strings as the result of mapping M_2) rather than strings of characters by calculating the number of insert, remove, and replacement operations to transfor vector $\mathbf{x}$ to vector $\mathbf{y}$, which is defined as $xform(\mathbf{x}, \mathbf{y})$. But should each type of transformation have the same weight? Isn't the replacement transformation, for example, comparable with a deleting procedure followed by an insertion procedure? Hence, we could argue that the cost function c should have the behavior $c(delete) + c(insert) \geq c(replace)$. We can then calculate the worst case transformation cost $xform_{wc}(\mathbf{x}, \mathbf{y})$ of $\mathbf{x}$ to $\mathbf{y}$ replacing all concept parts of $\mathbf{x}$ with parts of $\mathbf{y}$, then deleting the remaining parts of $\mathbf{x}$, and inserting additional parts of $\mathbf{y}$. The worst case cost is then used to normalize the edit distance resulting in

$$sim_{levenshtein}(R_x, R_y) = \frac{xform(\mathbf{x}, \mathbf{y})}{xform_{wc}(\mathbf{x}, \mathbf{y})} \qquad (8)$$

For the example in Figure 1, the minimum transformation cost from $\mathbf{x}$ to $\mathbf{y}$ is 3 (three replacements of weight 1). The worst case transformation cost is 9 which results in the final Levenshtein edit distance of 0.66 from vector $\mathbf{x}$ to $\mathbf{y}$.

3. iRDQL: EXTENDING RDQL WITH SIMILARITY JOINS

RDQL (RDF Data Query Language) [18, 19] is a query language to formulate queries over RDF [15] in Jena models [5]. It looks at the data stored in the model as RDF triples which consist of a subject, a property, and an object where the object may be again a resource or a literal respectively. For illustration purposes refer to Figure 2 that shows a fragment of an RDF-graph of a particular OWL-S [12] service instance (*Beach Surfing Service*). Given a model that holds this instance, the following RDQL query (shortened)

```
SELECT ?S1,?P1
WHERE  ?S1 presents ?P1
       ?P1 serviceName ‘‘beach surfing’’
       ?P1 textDescription ‘‘It returns ...’’
```

finds exactly the service instance *Beach Surfing Service* and its profile *Beach Surfing Profile*. The output produced by RDQL is depicted in Table 1.

Table 1: Output of simple RDQL query

S1	P1
Beach Surfing Service	Beach Surfing Profile

Suppose now you are not interested in finding exactly the instance *Beach Surfing Service* that presents the profile *Beach Surfing Profile* but in finding services that present profiles similar to *Beach Surfing Profile*. To achieve this goal we extended the RDQL language with three additional language constructs which we call `IMPRECISE`, `SIMMEASURE` and `OPTIONS`. The `IMPRECISE` clause defines the variables of the query whose bindings

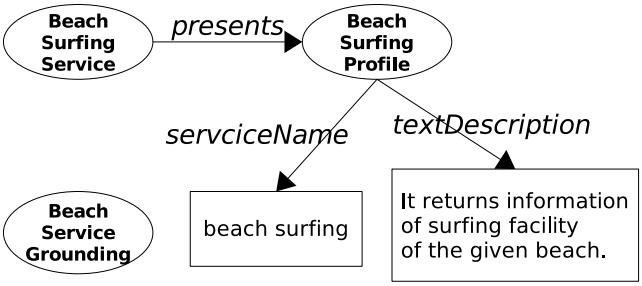


Figure 2: Partial semantic network representation of a concrete OWL-S service instance *Beach Surfing Service*

(found resources) should be matched imprecisely when executing the query. That is, they are added to the result set of the query together with their corresponding similarity value as computed by the similarity measure. The measure to compare two resources is specified by the `SIMMEASURE` clause. Here any similarity measure implemented in SimPack can be used. Additional parameters of the similarity measure can be specified by the `OPTIONS` clause. The extended iRDQL syntax looks as follows:

```
SELECT      [selectClauseVariables]
FROM        [fromClause]
WHERE       [whereClause]
AND         [filterClause]
USING       [usingClause]

IMPRECISE   [impreciseClauseVariables]
SIMMEASURE  [similarityMeasureClause]
OPTIONS     [similarityMeasureOptionsClause]
```

As an example consider that the three services *Beach Surfing Service, Beach Broker Service,* and *Abstract Broker Service* together with their corresponding service profiles *Beach Surfing Profile, Beach Broker Profile,* and *Abstract Broker Profile* are stored in a Jena model. The query corresponding to our users desiderata would look like the following:

```
SELECT ?S1,?P1,?P2
WHERE  ?S1 presents ?P1
       ?P2 serviceName ‘‘beach surfing’’
       ?P2 textDescription ‘‘It returns ...’’
IMPRECISE ?P1,?P2
SIMMEASURE Levenshtein
OPTIONS IGNORECASE false THRESHOLD 0.7
```

The query looks for a service `?S1` with profile `?P1`. It retrieves all profiles `?P2` that have the service name *“beach surfing”* and the textual description *“It returns ...”*.[2] The similarity between `?P1` and `?P2` is computed using the Levenshtein string edit distance and is returned with the possible combinations of `?S1`, `?P1`, and `?P2` as shown in Table 2. String comparison is case sensitive. A threshold of 0.7 is used in this example expressing that two strings are equal if their edit distance is at least 0.7.

4. EXPERIMENTAL EVALUATION

In order to assess the usefulness of iRDQL, we need to evaluate it by (1) choosing a knowledge domain and (2) specifying queries for which we (3) have a set of relevant answers. This way, a statistical analysis of the results of an iRDQL query can provide a sense of the utility of our approach. This section will describe the experimental setup and the statistical evaluation of the results, setting the stage for a discussion of the results in the next section.

4.1 OWL-S Test Collection

We chose the `OWL-S-TC-v1` [9] semantic web service retrieval test collection as the knowledge domain for our evaluation.[3] This collection specifies a set of 406 OWL-S services of six different domains (i.e., *communication, economy, education, food, medical,* and *travel*). The collection is intended to support the evaluation of the performance of OWL-S semantic web service matchmaking algorithms. For each domain, the collection specifies a number of queries along with a set of relevant answers. For instance, if querying for a service called *City Broker Service* that should provide the best hotel reservation service for a given city, the collection specifies 13 additional services (such as *City Financial Agent Service, City organization Service, etc.*) which are considered as relevant answers to this query.

4.2 Experimental Setup

The following steps were necessary to evaluate our approach: For each query, all OWL-S services of the same domain as the query are loaded into a Jena model. (Each service is specified in a separate OWL-S file and one by one loaded into the model.) Next, we (automatically) generated a proper iRDQL statement from the OWL-S query file.[4] For each iRDQL query we applied the mapping M_2 (see Section 2.2) to the resources that should be matched imprecisely and compared the resulting vectors of strings using the Levenshtein similarity measure as explained in Section 2.2. Both, the

[2]In our current approach `?P2` acts just like a *target* for comparison, i.e., all retrieved profiles `?P1` are compared against this target `?P2`.
Although not applied at the moment, additional query expansion methods can be used as, for instance, described in [23] to further extend the query's search space.

[3]The test collection is freely available as open source at `http://www.semwebcentral.org`.

[4]Refer to appendix A that shows a complete iRDQL query as produced by our implementation.

Table 2: Output of the extended iRDQL example query

S1	P1	P2	Sim
Beach Surfing Service	Beach Surfing Profile	Beach Surfing Profile	1.0
Beach Broker Service	Beach Broker Profile	Beach Surfing Profile	0.85
Abstract Broker Service	Abstract Broker Service	Beach Surfing Profile	0.7

OWL-S profile that is presented by a service and the service's process model are declared as imprecise. That is, when searching for a service with some profile P and model M the query is intended to find and rank all services that have a similar profile P' and model M'.[5] As final similarity value we chose the average of the similarities of the service profiles (P and any of P') and the process models (M and any of M').

4.3 First Results

In Figure 3.1, precision, recall, and f-measure [22] for the query *french_english_converterService* are depicted.[6] The test collection names 10 services out of 26 being relevant to this query. The numbers n on the x-axis express the first n top-ranked services of the result set of the executed iRDQL query. In other words, the number expresses the size of the result set of retrieved services, all of which are top-ranked and listed in descending order of their similarity to the service *french_english_converterService*.

In Figure 3.1, the precision of the iRDQL query is 1.0 for a result set of size one to eight indicating that all of the services inside that result set are considered as correct answers to the query service *french_english_converterService*. The overall trend of precision is decreasing since the result set is constantly growing until its size reaches the total number of services held in the model. The recall of the iRDQL query constantly increases as additional relevant services are added to the result set of the query. At a result set size of 14, the recall reaches 1.0 expressing that all relevant services are found and included in the result set.

The behavior observed in Figure 3.1 (as well as in Figures 3.2 to 3.6) illustrates the usefulness of our approach. In each of the domains, an exact query (i.e., an RDQL query with no similarity extension) yields exactly one result: the only perfect match. While this result has 100% precision it has a rather poor recall (10% for the query *french_english_convertorService*, similar results in other domains). The use of the imprecision extension for RDQL allowed us to simply extend the reach of the query and find additional correct matches without (at least initially) overly decreasing recall.

In addition, to further evaluate our approach we generated the Levenshtein-based iRDQL statement that applies the similarity join only to the *service profile* (called *iP*). Figure 4 shows the average precision, recall, and f-measure for each of the query styles (*iP* vs. *iPM*). Actually, as the comparison of both query styles shows, an increased use of similarity operators leads to better retrieval performance: *iPM* (which has two similarity joins) significantly outperforms *iP* (only one similarity join) on all measures as shown by a t-test (precision: $p = 1.4e^{-19}$, recall: $p = 9.8e^{-21}$, f-measure: $p = 5.7e^{-19}$).

Obviously, this evaluation of iRDQL can only serve as an illustration. A thorough evaluation will have to (1) compare the approach's robustness towards the use of different similarity measures and mappings, (2) explore different combination approaches for multiple similarity measures, (3) investigate the computational consequences of using similarity joins over precise joins, and (4) investigate the quality of queries constructed by developers in the light of these new constructs.

5. CONCLUSIONS AND FUTURE WORK

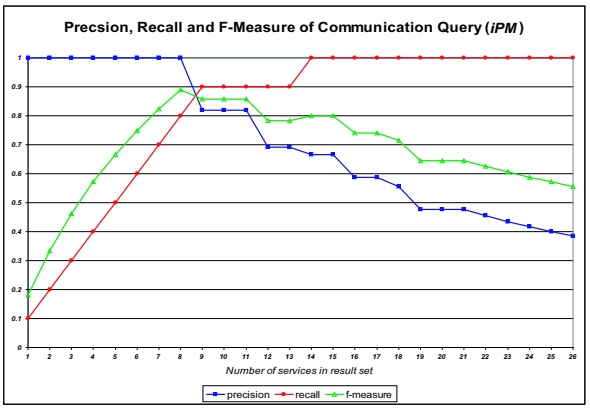
In this paper we introduced our approach of extending RDQL with similarity joins to find not only precise matches to a query but also a set of similar matches. For this purpose, iRDQL uses a similarity measure to determine the degree of similarity between two semantic web resources. Although we used only a single similarity measure, SimPack implements a number of measures, all of which can be used within iRDQL. Finding the right measure to determine the similarity between resources highly depends on the field of application and the format of the data to be compared. Thus, further research is necessary to find the best performing similarity measure for the combination of iRDQL and OWL-S semantic web service descriptions.

Our implementation was inspired by Cohen's approach [6] of using similarity joins to solve the problem of combining information of relations from different, heterogeneous databases. Cohen uses similarity joins to integrate such databases which are assumed to store textual information. He uses a standard *tf-idf* scheme [1] and the cosine measure (see equation 1) to compute the similarity between fragments of text. The difference to our

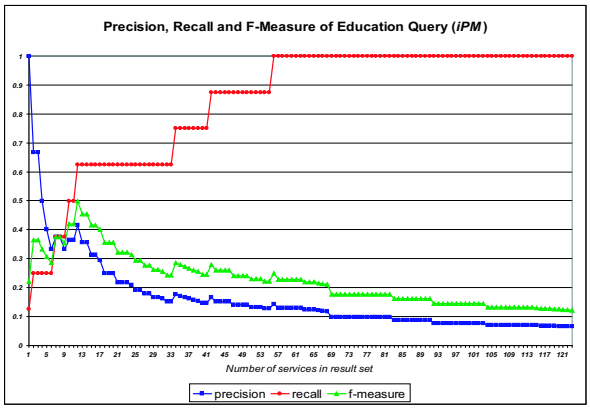
[5] We call this query style *iPM* when querying for both, similar profiles as well as similar process models, respectively.

[6] The f-measure combines recall and precision in a single efficiency measure (it is the harmonic mean of precision and recall).

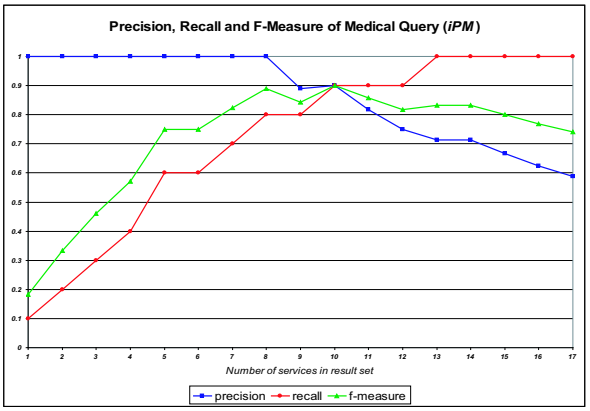
$$\text{f-measure} = \frac{2 * recall * precision}{recall + precision}$$



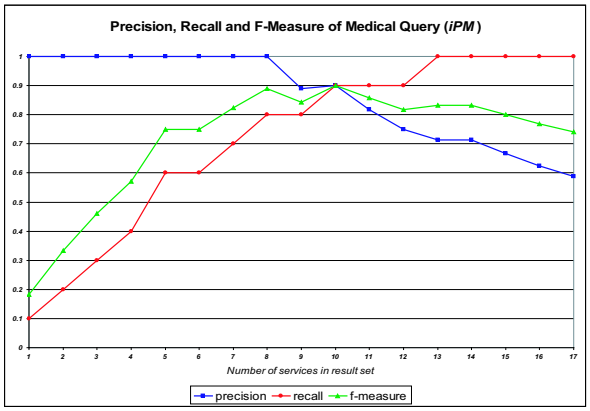
3.1: *french_english_convertorService*



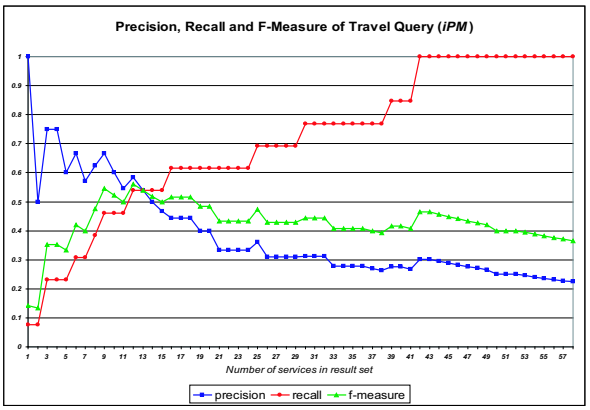
3.2: *university_address_service*



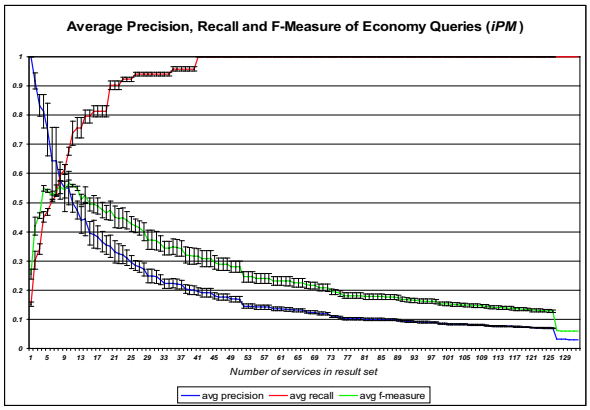
3.3: *foodfish_wine_service*



3.4: *covering_food_service*



3.5: *city_broker_service*



3.6: average precision, recall, and f-measure for the economy queries

Figure 3: The figure shows precision, recall, and f-measure for the 5 queries of the domains *communication* (3.1), *education* (3.2), *food* (3.3), *medical* (3.4), and *travel* (3.5). Figure 3.6 depicts the average precision, recall, and f-measure for the four queries of the domain *economy* (the variance of the average is illustrated by error bars).

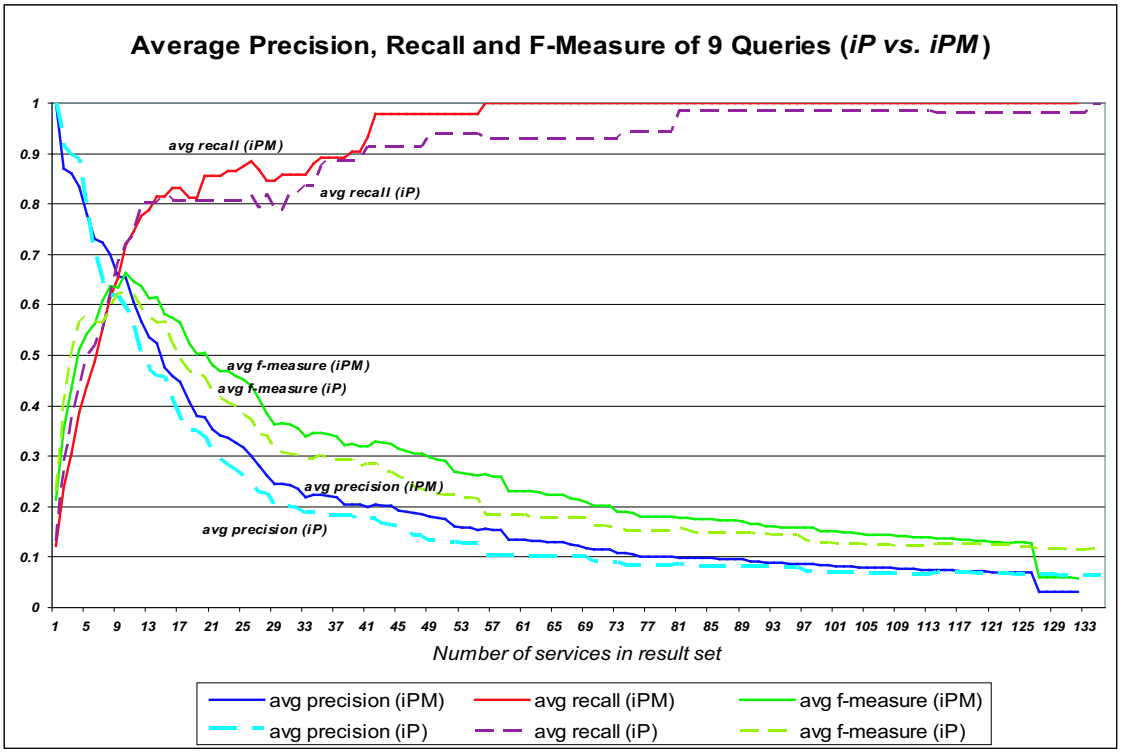


Figure 4: The figure shows the average precision, recall, and f-measure for the 9 queries of the collection *(iP vs. iPM)*. *iPM* significantly outperforms *iP* on precision, recall, and f-measure.

approach is that we are not dealing with flat tables (i.e., data stored in first normal form) but with complex (ontologized) objects (i.e., data stored in NF^2—non first normal form [17]). This calls for a deeper investigation of similarity measures that rely on the structure of an ontology. In accordance with Cohen's work we claim that the approach presented in iRDQL provides the basis for combining the strengths of logic-based precise querying and similarity-based retrieval.

6. REFERENCES

[1] R. Baeza-Yates and B. d. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.

[2] A. Bernstein, E. Kaufmann, C. Bürki, and M. Klein. How Similar Is It? Towards Personalized Similarity Measures in Ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*, February 2005.

[3] A. Bernstein, E. Kaufmann, and C. Kiefer. SimPack: A Generic Java Library for Similarity Measures in Ontologies. Technical report, University of Zurich, Department of Informatics. `http://www.ifi.unizh.ch/ddis/staff/goehring/btw/files/ddis-2005.01.pdf`, 2005.

[4] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[5] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: Implementing the Semantic Web Recommendations. Technical report, HP Labs, 2003.

[6] W. W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Trans. Inf. Syst.*, 18(3):288–321, 2000.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[8] D. Gentner and J. Medina. Similarity and the Development of Rules. *Cognition*, vol. 65:263–297, 1998.

[9] M. Klusch. OWLS-TC-v1: OWL-S Service Retrieval Test Collection. `http://projects.semwebcentral.org/projects/owls-tc/`, 2005.

[10] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, vol. 10:707–710, 1966.

[11] D. Lin. An Information-Theoretic Definition of Similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[12] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/, November 2004.

[13] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. http://www.w3.org/TR/owl-semantics/, February 2004.

[14] P.W.Lord, R. Stevens, A. Brass, and C.A.Goble. Investigating semantic similarity measures across the gene ontology: the relationship between sequence and annotation. *Bioinformatics*, 19(10):1275–83, 2003.

[15] RDF Core Working Group. RDF Primer. http://www.w3.org/TR/rdf-primer/, 2004.

[16] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[17] H. J. Schek and M. H. Scholl. The Relational Model With Relation-Valued Attributes. *Inf. Syst.*, 11(2):137–147, 1986.

[18] A. Seaborne. Jena Tutorial – A Programmer's Introduction to RDQL. http://jena.sourceforge.net/tutorial/RDQL/, 2004.

[19] A. Seaborne. RDQL – A Query Language for RDF. http://www.w3.org/Submission/RDQL/, 2004.

[20] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In *Pattern Matching Algorithms*, pages 341–371. Oxford University Press, 1997.

[21] A. Strehl, J. Ghosh, and R. Mooney. Impact of Similarity Measures on Web-page Clustering. In *Proceedings of the 17th National Conference on Artificial Intelligence: Workshop of Artificial Intelligence for Web Search (AAAI 2000), 30-31 July 2000, Austin, Texas, USA*, pages 58–64. AAAI, July 2000.

[22] C. van Rijsbergen. *Information Retrieval*. 2nd edition, 1979.

[23] E. M. Voorhees. Query Expansion Using Lexical-Semantic Relations. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 61–69, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

APPENDIX

A. COMPLETE iRDQL QUERY

```
SELECT ?s, ?p, ?p1, ?m, ?m1

WHERE (?s rdf:type sv:Service)
      (?s sv:supports ?g)
      (?g rdf:type gr:WsdlGrounding)
      (?g sv:supportedBy ?s)
      (?s sv:presents ?p1)
      (?p1 rdf:type pr:Profile)
      (?p1 sv:isPresentedBy ?s)
      (?s sv:describedBy ?m1)
      (?m1 rdf:type px:ProcessModel)
      (?m1 sv:describes ?s)
      (?p rdf:type pr:Profile)
      (?p sv:isPresentedBy ?s1)
      (?s1 rdf:type sv:Service)
      (?p pr:serviceName ?sn)
      (?p pr:textDescription ?sd)
      (?p pr:hasInput ?in1)
      (?p pr:hasOutput ?out1)
      (?in1 px:parameterType ?in1PT)
      (?in1 rdfs:label ?in1L)
      (?out1 px:parameterType ?out1PT)
      (?out1 rdfs:label ?out1L)
      (?m rdf:type px:ProcessModel)
      (?m sv:describes ?s2)
      (?s2 rdf:type sv:Service)
      (?m pr:hasProcess ?x)
      (?x rdf:type px:AtomicProcess)
      (?x px:hasInput ?in2)
      (?x px:hasOutput ?out2)
      (?in2 px:parameterType ?in2PT)
      (?in2 rdfs:label ?in2L)
      (?out2 px:parameterType ?out2PT)
      (?out2 rdfs:label ?out2L)

AND ?sn =~ /beach surfing/i
AND ?sd =~ /It returns information.../i
AND ?in1 =~ /_BEACH/
AND ?out1 =~ /_SURFING/
AND ?in1 eq ?in2
AND ?out1 eq ?out2

USING sv for <Service.owl#>
      pr for <Profile.owl#>
      px for <Process.owl#>
      gr for <Grounding.owl#>

IMPRECISE ?p, ?p1
IMPRECISE ?m, ?m1

SIMMEASURE Levenshtein
OPTIONS IGNORECASE false THRESHOLD 0.7;
```