Semantic Process Retrieval with iSPARQL

Christoph Kiefer[1], Abraham Bernstein[1], Hong Joo Lee[2], Mark Klein[2], and Markus Stocker[1]

[1] Department of Informatics, University of Zurich, Switzerland
{kiefer,bernstein,stocker}@ifi.unizh.ch
[2] Center for Collective Intelligence, Massachusetts Institute of Technology, USA
{hongjoo,m_klein}@mit.edu

Abstract. The vision of semantic business processes is to enable the integration and inter-operability of business processes across organizational boundaries. Since different organizations model their processes differently, the discovery and retrieval of similar semantic business processes is necessary in order to foster inter-organizational collaborations. This paper presents our approach of using iSPARQL– our imprecise query engine based on SPARQL– to query the OWL MIT Process Handbook– a large collection of over 5000 semantic business processes. We particularly show how easy it is to use iSPARQL to perform the presented process retrieval task. Furthermore, since choosing the best performing similarity strategy is a non-trivial, data-, and context-dependent task, we evaluate the performance of three simple and two human-engineered similarity strategies. In addition, we conduct machine learning experiments to learn similarity measures showing that complementary information contained in the different notions of similarity strategies provide a very high retrieval accuracy. Our preliminary results indicate that iSPARQL is indeed useful for extending the reach of queries and that it, therefore, is an enabler for inter- and intra-organizational collaborations.

1 Introduction

One of the cornerstones of the Semantic Web services vision is to enable the design and execution of dynamic inter- and intra-organizational services (processes). A major prerequisite for fulfilling this vision is the ability to find services which have certain features (*i.e.*, the ability for adaptive service discovery/-matchmaking and/or mediation). Most approaches so far have relied on some type of logical reasoning [4, 10]. In earlier works, we suggested that statistical methods based on a catalog of simple predefined similarity measures might be more suitable for this task [3]. Indeed, using the OWLS-TC matchmaking test collection, we showed that a straightforward method based on simple, off-the-shelf similarity metrics performed almost as well as the "best of bread" OWLS-MX matchmaker that was engineered to the task of matching OWL-S services.

While this success was remarkable it left open some important questions. First, the question of *which similarity measure* is applicable for a given problem needs to be answered. Findings from machine learning [8], information retrieval

[1], and psychology [9] show that the best performing similarity measure might be both task (*e.g.*, OWL-S/WSML matchmaking, retrieval in ontologies, etc.) and domain dependent (*i.e.*, the ontologies involved). Indeed, finding the best similarity measure for any given task and domain can be mapped to an optimization problem, where the "No Free Lunch" theorem [15] has proven that no uniformly best solution exists. Hence, the choice of the best performing similarity measure for any given task given an application domain seems anything but straightforward.

Second, given that our similarity-based approach was still slightly outperformed by the human-engineered, task-optimized OWLS-MX matchmaker, gives rise to the question *if a (human-) engineered, task-optimized similarity measure would not perform better*? This question is especially important since in many practical applications a considerable amount of human (knowledge) engineering is expended to improve the performance of systems. Hence, the engineering effort would also go into similarity-based solutions.

Third, given that similarity is an inherently statistics-based notion almost begs *the use of statistical machine learning techniques for finding a similarity measure optimized for a given task and application domain.*

In this paper we use the *iSPARQL* framework to address exactly these questions. iSPARQL is an extension of official SPARQL that allows for similarity joins which employ any of about 40 different similarity measures implemented in *SimPack*[3] – our generic Java library of similarity measures for the use in ontologies. It, therefore, lends itself as a platform for any kinds of similarity-based retrieval experiments in ontologies. Specifically, the contributions of this paper are that it (i) shows the simplicity of designing similarity based Semantic Web applications with iSPARQL, (ii) analyzes the usefulness of human-engineered task- and domain-specific similarity measures in comparison to some off-the-shelf measures widely used in computer science and AI, and (iii) shows how similarity measures learned through supervised learning techniques outperform both the off-the-shelf as well as the human-engineered measures in a service retrieval task. Last, the paper introduces a new data set for (process/service) retrieval applications in ontologies based on the MIT Process Handbook [13] that provides a very rich structural and textual description of the provided processes.

The remainder of this paper is structured as follows. The next section succinctly summarizes the most important related work. Given the importance as an underlying framework, Section 3 introduces the relevant features of iSPARQL. Section 4 is the heart of the paper: it introduces the experimental setup including the data set used in the evaluations, provides some details on the experiments, and discusses the results. To close, Section 5 discusses the results in the light of the claims, related work, and limitations. We close the paper with our conclusions and some insight into future work.

[3] `http://www.ifi.unizh.ch/ddis/simpack.html`

2 Related Work

Several other studies focus on the comparison of semantic business processes either for retrieval, discovery, matchmaking, or process alignment. We introduced in earlier works PQL – the Process Query Language to query the MIT Process Handbook [4]. PQL does not make use of similarity measures to retrieve similar query matches. However, PQL knows a "contains"-operator that can be roughly compared with the SQL "like"-operator performing string comparisons.

We are aware of two other studies that address the task of aligning semantic business processes using a similarity measure. Brockmans *et al.* and Ehrig *et al.* [5, 7] propose an approach to semantically align business processes originally represented as Petri nets. After the nets have been transformed to OWL, similarity measures from different categories are employed to measure the affinity between elements of Petri nets. Since we are able to define similarity strategies (*i.e.*, compositions of several atomic similarity measurements and weighting schemes) with our iSPARQL system, we consider such an ontology alignment task as being in the range of tasks which could be perfectly carried out by iSPARQL.

With respect to matchmaking, Klusch *et al.* [10] present an approach to perform hybrid Semantic Web service matchmaking. Their OWLS-MX matchmaker uses both, semantic similarity measures, as well as logic-based reasoning techniques to discover similar web services to a given query service. Again, Semantic Web service/process matchmaking is a possible application for iSPARQL.

Last, imprecise RDQL (iRDQL) is the predecessor of iSPARQL [3]. In iRDQL, special keywords are used to specify the similarity strategy (and parameters) to measure the relatedness between resources in ontologies. We did not want to introduce new keywords in iSPARQL since this would break the official W3C SPARQL grammar. Hence, we decided to integrate imprecise statements as *virtual triples* allowing us to add similarity measures and parameters by simply extending the virtual triple ontology.

3 iSPARQL

This section succinctly introduces the relevant features of our iSPARQL framework that serves as the technical foundation to all evaluations.[4] iSPARQL is an extension of SPARQL [14] that allows to query by triple patterns, conjunctions, disjunctions, and optional patterns. iSPARQL extends the traditional SPARQL grammar but does not make use of additional keywords. Instead, iSPARQL introduces the idea of *virtual triples*. Virtual triples are not matched against the underlying ontology graph, but used to configure similarity joins: they specify which pair of variables (that are bound by SPARQL to resources) should be joined and compared using what type of similarity measure. Thus, they establish a *virtual relationship* between the resources bound to the variables describing their similarity. A similarity ontology defines the admissible virtual triples

[4] An online demonstration of iSPARQL is available at `http://www.ifi.unizh.ch/ddis/isparql.html`

```
1  PREFIX ph: <http://www.ifi.unizh.ch/ddis/ph/2006/08/ProcessHandbook.owl#>
2  PREFIX isparql: <java:ch.unizh.ifi.isparql.query.property.>
3
4  SELECT ?process1 ?name1 ?overallsimilarity
5  WHERE {
6   ?process1 ph:name ?name1 .
7   ?process1 ph:description ?description1 .
8   ?process2 ph:name ‘‘Sell’’ ; ph:name ?name2 .
9   ?process2 ph:description ?description2 .
10
11  # ImpreciseBlockOfTriples (lines 13-20, 22-24, and 26-33)
12
13  # NameStatement
14  ?strategy1 isparql:name ‘‘LoLN’’.
15  # ArgumentsStatement
16  ?strategy1 isparql:argument (?name1 ?name2) .
17  # IgnorecaseStatement
18  ?strategy1 isparql:ignorecase ‘‘true’’ .
19  # SimilarityStatement
20  ?strategy1 isparql:similarity ?sim1
21
22  ?strategy2 isparql:name ‘‘TFIDFD’’ .
23  ?strategy2 isparql:arguments (?description1 ?description2) .
24  ?strategy2 isparql:similarity ?sim2 .
25
26  ?strategy3 isparql:name ‘‘ScoreAggregator’’ .
27  # ScoresStatement
28  ?strategy3 isparql:scores (?sim1 ?sim2) .
29  # WeightsStatement
30  ?strategy3 isparql:weights (0.8 0.2) .
31  # AggregatorStatement
32  ?strategy3 isparql:aggregator ‘‘sum’’ .
33  ?strategy3 isparql:similarity ?overallsimilarity
34 } ORDER BY DESC(?overallsimilarity);
```

Listing 1.1. iSPARQL example query for the MIT Process Handbook.

and links the different measures to their actual implementation in our library of similarity measures called *SimPack*. The similarity ontology also allows the specification of more sophisticated combinations of similarity measures, which we call *similarity strategies* (or simply *strategies*) in the rest of this paper. Note that the order of virtual triples is irrelevant since iSPARQL's query processor will inspect (reorder) them before the query is passed to the query engine. In the remainder of this section, we will briefly discuss the iSPARQL grammar and then introduce some of the similarity strategies employed in the evaluation.

3.1 The iSPARQL Grammar

The various additional grammar statements are explained with the help of the example query in Listing 1.1. This query aims at finding processes (or services) in a process ontology (we use the MIT Process Handbook introduced in Section 4.1) which are similar to the process "Sell" by comparing process names and descriptions. To implement our virtual triple approach we added an `ImpreciseBlockOfTriples` symbol to the standard SPARQL grammar expression of `FilteredBasicGraphPattern` [14]. Instead of matching patterns in the RDF graph, the triples in an `ImpreciseBlockOfTriples` act as *virtual triple* patterns, which are interpreted by iSPARQL's query processor

An `ImpreciseBlockOfTriples` requires at least a `NameStatement` (lines 14, 22, and 26) specifying the similarity strategy. iSPARQL has two kinds of strategies: *similarity strategies* and *aggregation strategies.* The former defines how the proximity of resources should be computed. The latter aggregates previously computed similarity scores to an overall similarity value. The example query in Listing 1.1 defines the two similarity strategies "LoLN" (lines 13–20) and "TFIDFD" (lines 22–24) as well as the aggregation strategy "ScoreAggregator" (lines 26–33; see Section 3.2 for a discussion of the available strategies).

In addition, an `ImpreciseBlockOfTriples` requires an `ArgumentsStatement` (lines 16 and 23) or a `ScoresStatement` (line 28), depending on whether it specifies a similarity or an aggregation strategy. An `ArgumentsStatement` specifies the resources under comparison to the iSPARQL framework. The `ScoresStatement` takes a list of previously calculated values (typically from similarity strategies) and summarizes the individual values in a user-defined way (*e.g.*, average, weighted sum, median, etc.). We found aggregators to be useful to construct overall (sometimes complex) similarity scores based on two or more previously computed similarity scores. The similarity ontology also allows the use of some additional triple patterns (statements) for most strategies to pass parameters to the strategies instructing them to, for example, ignore a string's case during a comparison operation (the `IgnorecaseStatement` on line 18) or to apply weights to the aggregated values (using the `WeightsStatement` on line 30).

3.2 Similarity Strategies

Currently, iSPARQL supports all of the about 40 similarity measures implemented in SimPack. The reference to the implementing class as well as all necessary parameters are listed in the iSPARQL ontology. It is beyond the scope of this paper to present a complete list of implemented strategies. Therefore, Table 1 summarizes the five similarity strategies we use to evaluate the performance of iSPARQL on the MIT Process Handbook (see Section 4). We distinguish between *simple* and *engineered* strategies: simple strategies employ a single, atomic similarity measure of SimPack, whereas engineered strategies are a (weighted) combination of individual similarity measures whose resulting similarity scores get aggregated by a user-defined aggregator. Table 1 lists in addition to the explanation of each strategy if it is considered as simple or engineered.

4 Experimental Analysis

The goal of our experimental analysis was to find some empirical evidence to answer the questions raised in the introduction: Which are the "correct" measures for a given task and domain? Do engineered measures outperform off-the-shelf measures? And, can an "optimal" measure be learned? To that end we

Strategy	Explanation
TFIDFD (simple)	TFIDF between process descriptions: the textual descriptions of two processes are compared by TFIDF, the standard information retrieval similarity measure. This measure makes use of pre-computed corpus of process descriptions which serves to retrieve statistics about words in the descriptions. The TFIDF measure extends the cosine measure with the traditional IR weighing scheme [1].
LevN (simple)	Levenshtein similarity of process names: two process names are compared with the Levenshtein string similarity measure [11]. The Levenshtein-based similarity measures are founded on the Levenshtein string edit distance that measures the relatedness of two strings (process names) in terms of the number of insert, remove, and replacement operations to transform one string into another string.
LoLN (simple)	Levenshtein Level 2 (Levenshtein of Levenshtein) similarity of process names: two process names such as "Buy over the internet" and "Sell via Internet" are compared string-by-string with the (inner) Levenshtein string similarity measure. If the similarity between two strings is above a user-defined threshold, the strings are considered as equal (*i.e.*, they match). These scores are used by the outer Levenshtein string similarity measure to compute an overall degree of similarity between the two process names (sequences of strings).
MITPH-LoLNTFIDFD (engineered)	Levenshtein Level 2 similarity between process names, TFIDF between process descriptions: this strategy is a combination of two atomic measures. An overall similarity score is computed by aggregating the individual scores.
MITPH-LoLNTFIDFD-JaccardAll (engineered)	Levenshtein Level 2 similarity between process names, TFIDF between process descriptions, Jaccard (Tanimoto) set-based similarity [6] between process exceptions, goals, resources, inputs, and outputs: a combination of six atomic measures; in addition to MITPH-LoLNTFIDFD, four single similarity scores are computed from two processes' goal, exception, resource, in- and output sets. An overall score is, again, determined by accumulating (and weighting) the individual scores.

Table 1. Selection of five iSPARQL similarity strategies.

constructed a large ontology retrieval data set and performed two sets of experiments: the pure *retrieval experiments* show a comparison of both off-the-shelf and domain/task-specific, engineered similarity strategies using iSPARQL; The *machine learning experiments* compare the performance of these predefined measures to learned strategies gained using supervised learning approaches. In the following, we first describe the experimental setup, explain the generation of the test set that is used to perform the aforementioned experiments, and present the results of our evaluations.

We conducted all our experiments on a two processor dual core AMD Opteron 270 2.0GHz machine with 4GB RAM, 7200rpm disks, using a 32Bit version of Fedora Core 5.

4.1 Test Set Generation – "Mutating" the MIT Process Handbook

In order to evaluate our ontology retrieval approach, we needed a substantial database of instances, which includes a sizable number of queries with its associated correct answers. The correct answers are crucial, as they allow the quantitative evaluation of the retrieval approach. But preparing manually a suitable database that is large enough to enable statistical analysis can be impracticably time-consuming. We, therefore, decided to bootstrap the data set generation process by a large existing knowledge base that describes business processes:

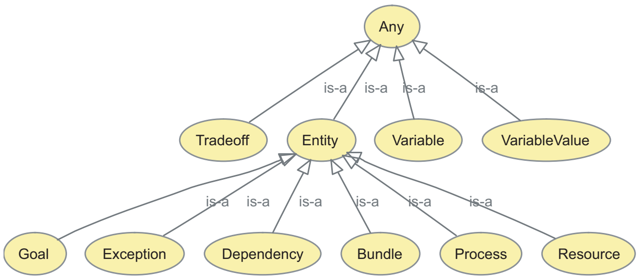


Fig. 1. Simplified structure of the OWL MIT Process Handbook Ontology.

The *MIT Process Handbook* is an electronic repository of best-practice business processes and the result of over a decade of development by over 40 researchers and practitioners centered around the MIT Center for Coordination Science.[5] The Handbook is intended to help people: (1) redesigning organizational processes, (2) inventing new processes, and (3) sharing ideas about organizational practices [13]. The Handbook includes a database of about 8000 business processes in addition to software tools for viewing, searching, and editing the database contents [12]. The Process Handbook is a process ontology: it provides a specialization hierarchy of processes (verbs) and their interrelationships in the form of properties, which connect the process to its attributes, parts, exceptions and dependencies to other processes. Note that specialization in the process handbook is non-monotonic. In other words, it is possible for a "child" process to overwrite or delete an inherited property. The Process Handbook, thus, has the advantage of being a sizable data set that was developed in a real-world setting (*i.e.*, by end-users and not by Semantic Web researchers).

In order to use the MIT Process Handbook for an evaluation, we had to *export it into an OWL-based format.* Given the non-monotonic inheritance structure the straight-forward translation of processes to concepts was not possible. We, therefore, decided to model the Process Handbook meta-model in OWL and export the processes in the Handbook as instances of the meta-model.[6] Hence, all major parts of the Handbook such as Process, Bundle, Goal, Exception, Resource, Dependency, and Trade-offs are represented as OWL classes (see Figure 1). With the ontology, we transformed the approximately 5000 business processes to OWL and stored them in their own files. Figure 2 shows a representative example of such a process.

Next, we had to find a sizable number of realistic queries and their corresponding correct answers in the Process Handbook. To that end *we adopted a novel approach for creating a test database that is based on semantics-preserving process mutation.* We began by *selecting 105 distinct process models* from within

[5] Now called the MIT Center for Collective Intelligence (`http://cci.mit.edu`).

[6] In order to preserve the inherent semantics of the MIT Process Handbook, some additional rules in RuleML would be needed [2].

```
@prefix rdfs:            <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:            <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:             <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:             <http://www.w3.org/2002/07/owl#> .
@prefix processHandbook: <http://www.ifi.unizh.ch/ddis/ph/2006/08/ProcessHandbook.owl#> .

<http://www.ifi.unizh.ch/ddis/ph/2006/08/E1024.owl#E1024> a processHandbook:Process ;
processHandbook:name "Determine cost" ;
processHandbook:description "This is a general activity to determine the cost
                             to the organization of purchase or production." ;
processHandbook:hasException <http://www.ifi.unizh.ch/ddis/ph/2006/08/E17159.owl#E17159> ;
processHandbook:hasSpecialization <http://www.ifi.unizh.ch/ddis/ph/2006/08/E6302.owl#E6302> ,
                                  <http://www.ifi.unizh.ch/ddis/ph/2006/08/E8007.owl#E8007> ;
processHandbook:hasGeneralization <http://www.ifi.unizh.ch/ddis/ph/2006/08/E3356.owl#E3356> .
```

Fig. 2. The figure shows process E1024 ("Determine cost") in Notation 3. E1024 has one exception, two process specializations, and one process generalization.

the Process Handbook repository. These models represent the target set. For each target process we then *created 20 variants of that process that are syntactically different but semantically equivalent* using mutation operators. These variants represent the "true positives" or correct answers (*i.e.*, the database items that should be returned when our retrieval algorithm is applied to find matches for the target process). All other items in the database are viewed as non-matches, and should not be returned by our retrieval algorithm if is it operating correctly.

Variants were created by applying semantics-preserving mutation operators to the target processes. Every variant represented the application of between 1 and 20 randomly selected operators to a target process. We used the following operators:

- a *process step* (*i.e.*, part of a process) is
  - split into two siblings (`STEPSPLIT`)
  - split into a parent/child (`STEPCHILD`)
  - merged with a (randomly selected) sibling (`STEPMERGESIB`)
  - merged with its parent (`STEPMERGEPARENT`)
  - deleted (`STEPDELETE`)
- a *word in the name* of a process is
  - deleted (`NAMEDELETE`)
- a *word in the description* of a process is
  - deleted (`DESCRIPTIONDELETE`)

The mutation operators were selected so that they produce a plausible alternative way of modeling the process they were applied to. If we were modeling a restaurant process, for example, some people might combine the "order" and "pay" actions into one substep (*e.g.*, for a fast food restaurant), while others might model the same process with separate substeps for "order" and "pay". These two approaches represent syntactically different, but semantically equivalent, ways of modeling the same process. The `STEPMERGESIB` operator could take a process model with distinct "order" and "pay" substeps and merge them into one. Conversely, the `STEPSPLIT` operator could take a process model where "order" and "step" are merged, and split them into two distinct substeps.

It should be noted, as a caveat, that there is a substantial random element in how the mutation operators work, since they do not perform a sophisticated semantic analysis of a step before, for example, deciding how to perform a split. Hence, the process variants may not look much like what a human might have generated, even though they are generated by a process that is similar to what a person might have used. It is our belief, however, that a semantics-preserving mutation approach represents a promising way for generating large query collections enabling rapid and useful evaluations of different retrieval algorithms. The algorithms that "rise to the top" as a result of this screening procedure can then be evaluated using hand-generated test sets that, presumably, will produce retrieval and precision figures that are closer to what we can expect in "real-world" contexts. The generated process retrieval test collection, including queries and variants (true positives) is available at our project web site.[7]

4.2 Retrieval Experiments – Off-the-shelf vs. Engineered

In order to compare the performance of off-the-shelf versus specifically engineered similarity strategies, we first chose three simple strategies from SimPack: TFIDFD, LevN, and LoLN (see Section 3.2). Obviously, we did not choose them randomly but actually chose the off-the-shelf measures that we thought would perform well and then discarded the ones that were not performing sufficiently well to compete with the top-ranking ones. Second, we manually defined (or engineered) two task and domain specific complex similarity strategies that are both a combination of multiple similarity measures based on our experience with the Process Handbook: MITPH-LoLNTFIDFD and MITPH-LoLNTFIDFDJaccardAll. Note that while almost no domain knowledge is necessary to choose and define the off-the-shelf similarity strategies, some domain expertise is needed for the human-engineered strategies since specifying which measures should be used to determine the similarity between which elements of processes means to have a profound understanding of the structure of the data.

To compare the performance of the similarity strategies, we had to execute all 105 query processes with each of the five similarity strategies. Here, the capabilities of iSPARQL were very useful: since it was designed to run SPARQL queries with similarity joins, we could simply construct iSPARQL queries that would correspond to the retrieval operations. Consider the query depicted in Listing 1.2: it computes a similarity join between the process with the reference http://www.ifi.unizh.ch/ddis/ph/2006/08/ProcessHandbook.owl#E16056 and all other entries in the knowledge base using the TFIDFD strategy and returns them ordered descending by similarity. Actually, we were able to run all five strategies with one query by having an `ImpreciseBlockOfTriples` (see Section 3.1) for each strategy in the same query, exemplifying how iSPARQL simplifies the implementation of Semantic Web retrieval applications.

To evaluate the performance of the queries we chose the traditional information retrieval measures precision and recall. As a representative example, the

[7] `http://www.ifi.unizh.ch/ddis/ph-owl.html`

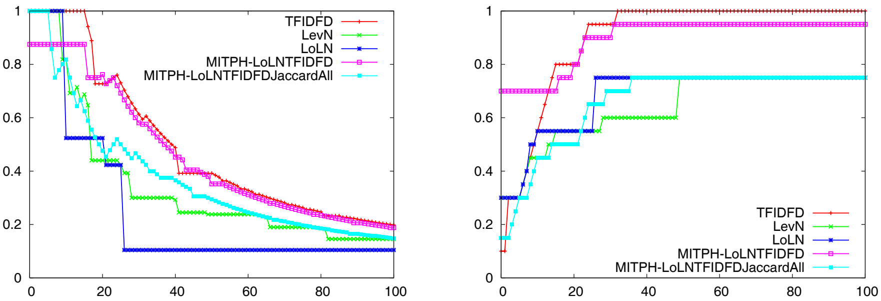
```
PREFIX ph: <http://www.ifi.unizh.ch/ddis/ph/2006/08/ProcessHandbook.owl#>
PREFIX isparql: <java:ch.unizh.ifi.isparql.query.property.>

SELECT ?process2 ?name2 ?similarity
WHERE {
 ?process2 ph:name ?name2 .
 ?strategy isparql:name ‘‘TFIDFD’’ .
 ?strategy isparql:arguments (ph:E16056 ?process2) .
 ?strategy isparql:similarity ?similarity .
ORDER BY DESC(?similarity)
```

Listing 1.2. iSPARQL retrieval query.

results for process E16056[8] are shown in Figures 3(a) and 3(b), which depict precision and recall for the 100 most similar processes to the query process. As one can see, TFIDFD outperforms all other strategies in terms of precision closely followed by MITPH-LoLNTFIDFD. Both, simple as well as engineered strategies start very high with precision=1, except for MITPH-LoLNTFIDFD that starts around 0.9. LoLN rapidly falls below 0.2 in precision (∼25 returned processes), which expresses its low usefulness for this retrieval task. Considering recall (Figure 3(b)), MITPH-LoLNTFIDFD starts highest (recall ∼0.7) but gets outperformed by TFIDFD (around 15 returns) for larger query result sets. Why does the standard TFIDF perform so well? We believe it is due to the large descriptions that are typically associated with Process Handbook entries. Given that the descriptions were not mutated in all cases and that mutation did essentially consist of deleting words, TFIDF, which has been found to be very useful in full text retrieval, may have an unfair advantage. Nonetheless, even disregarding TFIDF as a competitor, it is interesting to observe that neither of the engineered measures uniformly outperforms the off-the-shelf ones in terms of precision, but that they only gain with larger result sets. Why does the engineered measure MITPH-LoLNTFIDFDJaccardALL not perform equally well (LoLN initially outperforms it in terms of precision and almost uniformly outperforms it in terms of recall)? This might be due to badly chosen weights of the individual similarity strategies (*i.e.*, instead of giving the same weights to TFIDFD, LoLN, and Jaccard, TFIDFD should probably be weighted much higher as indicated by the simple strategies). We discuss an approach of how to learn such weights in the next subsection.

Figure 4 shows average precision and recall of the five employed similarity strategies across all 105 queries. As the figure illustrates the performance of all measures across all the queries is not as good as for the the single query. Nonetheless, we can see that the findings from the one query generalize qualitatively. Specifically, Figure 4(a) illustrates that the simple TFIDFD measure clearly outperforms all other strategies in terms of precision – it seems as if the descriptions across all the queries again are of much higher importance than other structure properties of a process. However note, that precision for all measures (including TFIDFD) on average is not as high as in the single query case.

[8] http://www.ifi.unizh.ch/ddis/ph/2006/08/E16056.owl



(a) Precision for E16056.

(b) Recall for E16056.

Fig. 3. Precision and recall for a representative example process.

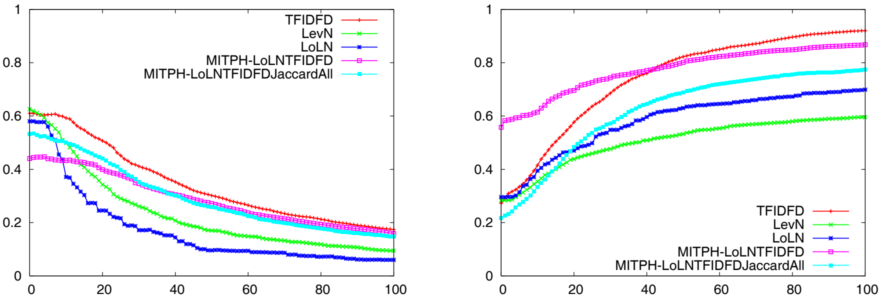
This due to the fact, that there are processes in the test collection which have shorter textual descriptions and/or fewer properties resulting in lower TFIDF similarity scores, which, in turn, leads to reduced average precision. In terms of *precision*, all three simple strategies outperform the engineered ones again for few processes returned. For larger query result sets, the two engineered strategies MITPH-LoLNTFIDFD and MITPH-LoLTFIDFDJaccardAll perform better than LevN and LoLN but still worse than simple TFIDF. Inspecting *recall* (Figure 4(b)), the best performing similarity strategy is the engineered MITPH-LoLNTFIDFD until ∼40 returned processes. With larger result sets, it gets outperformed by TFIDFD that starts with about the recall of low (recall ∼0.3). We note, that also on average, similarity strategies incorporating TFIDF to measure the relatedness of processes of the MIT Process Handbook perform substantially better than strategies focusing on other modeling aspects. Thus, future strategies should probably use TFIDF as one of their component measures, assigning it a high enough weight in the overall similarity computation.

Summarizing, we can state, that the engineered measures do not uniformly outperform the off-the-shelf ones. Indeed, it seems that the simple ones that are heavily reliant on full-text (such TFIDF) are favored by this data set. Ignoring the description (and the TFIDF measure), however, we can see that the engineered measures perform better in terms of both precision and recall for large return sets. For small return sets the off-the-shelf measures are better in terms of precision and at least competitive for recall.

4.3 ML Experiments – Off-the-shelf and Engineered vs. Learned

The last question raised in the introduction demands clarification on the performance on a learned measure in comparison to either the off-the-shelf or the engineered ones. To that end we decided to employ the widely used machine learning tool Weka[9] in conjunction with iSPARQL to learn a similarity measure based on

[9] http://www.cs.waikato.ac.nz/ml/weka/



(a) Average precision.

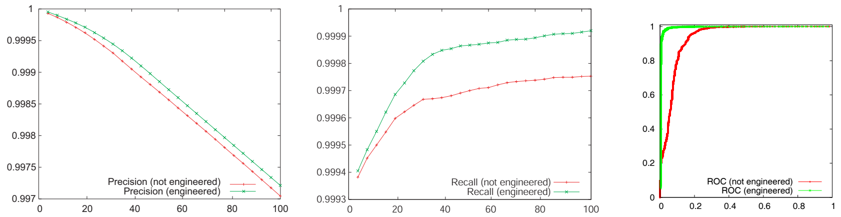
(b) Average recall.

Fig. 4. Average precision and recall for 105 queries and five similarity strategies.

the results obtained with the simple as well as engineered strategies. Specifically, for each of the 105 queries we took all the off-the-shelf but the TFIDF measures used so far. The rationale for not using the TFIDF measure was that we did not want the description to have too much influence in this evaluation. Together with the information if they were a correct or incorrect answer, we combined them to a feature vector $\mathbf{shelf_{i,j}} = [LevN(i,j), LoLN(i,j), correct(i,j)]^T$, where i is the number of the target entity, j is the number of the entity from the Process Handbook and $correct(i,j)$ specifies if j is a correct answer to the query i. We then combined all the vectors $\mathbf{shelf_{i,j}}$ to the data set *shelf*. Analogously, we constructed the vector $\mathbf{engineered_{i,j}}$ that extended $\mathbf{shelf_{i,j}}$ with the engineered measures to the data set *engineered*.

For each of these two data sets we then learned a similarity measure using a logistic regression statistical learning algorithm performing an (almost) 10-fold cross validation.[10] We took 10% of the queries (always exactly 10, discarding the rest), learned the similarity measure using the logistic regression learner on the remaining 90% of the data, and then measured its effectiveness on these 10%. This approach is standard practice in machine learning. The averages of the results of the 10 runs are shown in Figure 5. As the Figures 5(a) and 5(b) show, the performance of the learned measures vastly outperforms both the engineered and the off-the-shelf measures (note the scale on the figures!). It, thus, seems that each of the measures employed contains some latent (potentially different) information about the similarity between the queries and its correct answers. Combined, they provide an excellent performance. Note also, that the similarity measure learned from the *engineered* data set (the upper line in Figures 5(a) and 5(b)) significantly outperforms the one learned from the *shelf* data set (lower curves). Since precision/recall curves are sometimes misleading when evaluating the performance of learning approaches, we also supply the average receiver operating characteristic (ROC) curves for both learned measures. The ROC

[10] We call it “almost” 10-fold cross validation because 105 queries cannot be divided into 10 equally sized groups, but 5 groups of 10 and 5 groups 11 queries.



(a) Precision curves. (b) Recall curves. (c) ROC curves.

Fig. 5. Results for the learned similarity measure (logistic regression).

curve graphs the true positive rate (y-axis) against the false positive rate (x-axis), where and ideal curve would go from the origin to the top left (0,1) corner, before proceeding to the top right (1,1) one. As Figure 5(c) clearly shows, the similarity measure learned from the *engineered* data set almost perfectly mimics a perfect prediction resulting in an accuracy of 99.469%; the one for the *shelf* data set being not much worse with an accuracy of 98.523%.

5 Discussion, Limitations, and Future Work

The findings of the preceding analysis are relatively clear. First, the ease of use of our iSPARQL framework for the presented semantic process retrieval task has been clearly shown. Evaluations, that previously would have had to be programmed tediously, could be effectuated by simply compose a query. The seamless integration of simple, off-the-shelf as well as human-engineered similarity strategies significantly simplified the implementation. We, therefore, believe that a declarative query language containing statistical reasoning elements like iSPARQL can significantly simplify the design and implementation of Semantic Web applications that include some element of similarity. Since such elements are included in many of the core Semantic Web applications (*e.g.*, matchmaking, retrieval in ontologies, ontology alignment, etc.), *tools such as iSPARQL can play an important role in simplifying the spread of the Semantic Web.*

Second, as our retrieval experiments showed, the human-engineered measures performed constantly better on large sets of processes than the off-the-shelf measures did. In contrast, the simple, off-the-shelf strategies turned out to be superior for smaller sets. Furthermore, strategies including the TFIDF measure, which heavily drew on the process descriptions performed better in terms of precision and recall. This indicates that the *off-the-shelf methods captured a different notion of the similarity between processes than the engineered ones.* This finding is further supported by the learned similarity measures. As Figure 5 shows the results of the algorithm learned with only the off-the-shelf data is somewhat less precise than the one learned with both the engineered and off-the-shelf methods. Hence, the information contained in the engineered measures is at least partially complimentary to the information contained in the off-the-shelf ones, which the learning algorithm can exploit. Arguably, this additional information is *the latent experience of the experts that was embedded in the engineered measures.*

Third, the learned measures clearly outperformed the designed or off-the-shelf ones. The learning algorithm's ability to combine the complimentary information contained in the different notions of similarity proved to provide an overall almost overwhelming accuracy. We can, therefore, clearly conclude that *the value of using learned similarity measures seems immense*, assuming that a sufficient number of examples is available: irrespective of whether we used off-the-shelf or expert measures, the learned measures performed close to perfect.

One major *limitation of our work is the choice of experimental data.* The generalizability of our findings across tasks and domains is limited by the fact that we (i) only used one data set, (ii) that this data set employed some generated data, (iii) we only ran one task, and (iv) that the test suite generation strategy might have influenced the results. Nonetheless, we believe that our findings are likely to hold across domains and task: First, extrapolating from information retrieval, where the choice of good similarity measures seem to permeate across both tasks and domains. Second, while our data set is not ideal, *it is one of the first ones* in the Semantic Web that contains a large data set with both queries and associated true answers. Such data sets are very costly to design and only their introduction to the community will allow comparative studies, which, ultimately, is the basis of science. Last, even though the true positives where generated (note that the data base itself was collected by domain experts), their generation process was guided by many years of experience with the type of data under study. We, therefore, believe that our findings will generalize at least across domains and possibly, given the ubiquity of similarity measures in computer science and AI, even across tasks.

We see a couple of *future research directions*: (1) extending our evaluation to other domains and tasks to ensure our findings' generalizability; (2) applying iSPARQL to different Semantic Web tasks such as service matchmaking and ontology alignment to shed some more light on its potential as a framework; and (3) investigating extensions to iSPARQL that will further improve its usefulness for additional tasks.

6 Conclusions

Our study investigated the use of similarity measures in a process ontology retrieval task using the iSPARQL framework. We found that the declarative nature of iSPARQL did significantly simplify the task prompting us to a deeper investigation of the applicability of iSPARQL to different Semantic Web tasks such as matchmaking and ontology alignment, beyond the presented process retrieval task. We also found that the combination of different notions of similarity string learning approaches significantly boosted the overall task performance. Therefore, as seen from our evaluations, the use of statistics, either directly employed by similarity strategies or by statistical learning algorithms, proved crucial for the performance in this task. For the Semantic Web in general, these findings raise the question whether the more wide-spread use of statistical reasoning elements would not improve the overall performance of its tools and applications.

Acknowledgment One of the authors was supported by the Korea Research Foundation Grant of the Korean Government (KRF-2006-214-D00193). We would like to thank the anonymous reviewers for their valuable comments.

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
2. A. Bernstein, B. Grosof, and M. Kifer. Beyond Monotonic Inheritance: Towards Non-Monotonic Semantic Web Process Ontologies. In *W3C Ws. On Frameworks for Semantics in Web Services*, 2005.
3. A. Bernstein and C. Kiefer. Imprecise RDQL: Towards Generic Retrieval in Ontologies Using Similarity Joins. In *Proc. of the 2006 ACM Symp. on Applied Computing (SAC '06)*, pages 1684–1689, New York, NY, 2006.
4. A. Bernstein and M. Klein. Towards High-Precision Service Retrieval. In *Proc. of the 1st Int. Semantic Web Conf. on The Semantic Web (ISWC '02)*, pages 84–101, London, UK, 2002.
5. S. Brockmans, M. Ehrig, A. Koschmider, A. Oberweis, and R. Studer. Semantic Alignment of Business Processes. In *Proc. of the 8th Int. Conf. on Enterprise Information Systems (ICEIS '06)*, pages 191–196, Paphos, Cyprus, 2006.
6. W. W. Cohen, P. Ravikumar, and S. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proc. of the IIWeb Ws. (IJCAI '03)*, 2003.
7. M. Ehrig, A. Koschmider, and A. Oberweis. Measuring Similarity between Semantic Business Process Models. In *Proc. of the 4th Asia-Pacific Conf. on Conceptual Modelling (APCCM '07)*, Ballarat, Victoria, Australia, 2007. to appear.
8. L. Geng and H. J. Hamilton. Interestingness Measures for Data Mining: A Survey. *ACM Comp. Surv.*, 38(3), 2006.
9. D. Gentner and J. Medina. Similarity and the Development of Rules. *Cognition*, 65:263–297, 1998.
10. 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 (AAMAS '06)*, pages 915–922, New York, NY, 2006.
11. V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, 1966.
12. T. W. Malone, K. Crowston, and G. A. Herman. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, Cambridge, MA, 2003.
13. T. W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for Inventing Organizations: Towards a Handbook of Organizational Processes. *Management Science*, 45(3):425–443, 1999.
14. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.
15. D. Wolpert and W. Mcready. No Free Lunch Theorems for Optimization. *IEEE TOEC*, 1(1):67–82, 1997.