Feedback-based Data Set Recommendation for Building Linked Data Applications

Hélio Rodrigues de Oliveira
Center for Informatics - Federal University of Pernambuco
Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária, 50.740-560 - Recife – PE, Brazil
+55 81 2126.8430
hro@cin.ufpe.br

Alberto Trindade Tavares
Center for Informatics - Federal University of Pernambuco
Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária, 50.740-560 - Recife – PE, Brazil
+55 81 2126.8430
att@cin.ufpe.br

Bernadette Farias Lóscio
Center for Informatics - Federal University of Pernambuco
Av. Jornalista Anibal Fernandes, s/n - Cidade Universitária, 50.740-560 - Recife – PE, Brazil
+55 81 2126.8430
bfl@cin.ufpe.br

ABSTRACT
The huge and growing volume of linked data is increasing the interest in developing applications on top of such data. One of the distinguishing features of linked data applications is that the data could come from any RDF data set available on the Web. Different from conventional applications, where the data sources are under control of the application's owner or developer, linked data applications follow the Semantic Web vision of a world full of reusable data. Considering a potentially large number of data sets, one of the primary challenges facing the development of such solutions is the identification of suitable data sources, i.e., data sets that could give a good contribution to the answer of user queries. In this paper, we discuss this problem and we present a feedback-based approach to incrementally identify new data sets for domain-specific linked data application.

Categories and Subject Descriptors
H.4 [**Information System Applications**]: Miscellaneous;
H.2 [**Database Management**]: Miscellaneous

General Terms
Algorithms, Management, Measurement, Experimentation.

Keywords
Semantic Web, Linked Data, Feedback.

1. INTRODUCTION
Applications built on top of linked data may offer generic functionalities as, for example, linked data browsers and search engines or may offer more domain-specific functionalities by accessing and integrating data from various linked data sets [3]. One of the primary challenges facing the development of domain-specific applications is the identification of relevant data sets. Considering a potentially large number of datasets, to manually identify suitable ones for a given application may become an unfeasible task.

In this paper we restrict our attention to applications whose aim is combining data from different data sets to cover the needs of specific user communities. We are interested on applications that follow a query federation pattern, where complex queries may be posed directly to a fixed set of data sources without creating local data sources replicas [3]. One of the main challenges of such approach is that performance problems may arise if the number of data sets becomes too large. Therefore, the number of data sets that an application intends to use must be controlled in such a way that just the more relevant data sets should be considered.

In this paper, we propose a user feedback-based approach to assist developers to find proper data sources while they are building domain specific linked data applications. Our approach involves two main tasks: i) to find a subset of data sets that answers the application queries and ii) to choose the most important data sets in this subset. The first phase focuses on filtering the huge volume of linked data sets available on the Web, while the second one ranks the candidate data sets obtained in the first phase in order to identify the best ones. One distinguishing issue of our approach is that instead of using keyword search to identify candidate data sets we consider the semantics of SPARQL queries to be posed or already submitted to the application. Moreover user feedback is used as a way to assess the relevance of the candidate data sets.

The remainder of this paper is organized as follows. Section 2 presents some examples to motivate the proposed approach. Section 3 introduces some preliminary definitions used as the basis for our proposal. Section 4 describes our approach while Section 5 discusses some initial experiments we performed to test our approach. Section 6 shows the related works. Section 7 presents conclusions and future work.

2. MOTIVATION
One of the main problems with building applications that make use of data sources available on the Web consists in finding relevant data sources. In a general way, a data source is considered relevant when contributes for answering queries posed to the application [5]. However, it may happen that a data set may contribute for answering an application query but the obtained answer does not meet the user requirements. This may occur because the data source has generic data and the user wants more specific data, for example, or the data set has data of poor quality, i.e., the data may be outdated or incorrect. In such cases, it is not enough just finding data sets that can answer the application queries it is also necessary to check if the available data meet the user requirements.

I-SEMANTICS 2012, 8th Int. Conf. on Semantic Systems, Sept. 5-7, 2012, Graz, Austria

Suppose, for example, an application, which aims at publishing information about academic researchers in Computer Science. One relevant query for such application could be: *Return all researchers who published papers in 2012*. Considering that the application is specific for Computer Science, it is not worth mentioning the researchers' area when formulating the query. Therefore, in this case, any data set that has bibliographic information about scientific papers could be considered relevant for its answer. This situation happens because just considering the application query, as criteria for selecting relevant data sets, is not enough, once that application queries may be generic and do not reflect precisely the user requirements. In this case, considering application queries as the only criteria for finding relevant data sets will lead to generic information too. Specifically, the application is interested in data sets as DBLP RKBExplorer[1] or DBLP L3S[2] that stores information about Computer Science bibliography. However, data sets like DBPedia[3] and Geonames[4] will also be considered relevant once that they have bibliographic information about the most famous researchers. In a similar way, PubMed[5] and RAE[6] are considered relevant because they have bibliographic information about medicine researchers and researchers working in UK institutions, respectively.

Suppose another scenario where an application offers information about songs in general, which are obtained from linked data sets available on the Web. Considering that this is a very popular domain then we suppose the existence of a lot of data sets and, consequently, there is a big probability of finding data sets of poor quality. In this context, even when application queries are more specific it is possible finding data sets that are able of answering the application queries but the retrieved data is not relevant from the user perspective.

In both cases we can conclude that considering just application queries is not sufficient to identify relevant data sources for a given application. Information about the contents of the data set is also necessary in order to assure that the data set meets the user requirements and, therefore, should be considered as a relevant one. In our approach we consider both criteria: i) *a data set filter criteria*: given the increasing number of data sets becoming available on the Web, there is a need for mechanisms to help filtering data sets according to a given domain and a specific set of data requirements, and ii) *a data set relevance criteria*: considering that the data sets may have poor quality, i.e., the data can be incorrect, incomplete or outdated, or may be very general (or very specific) for a given application, the need arises for solutions that help to identify relevant data sets according to the user requirements.

In our approach, we use application queries for data sets filtering and user feedback as way of capturing the relevance of a data set. Application queries are a good start pointing for identifying information that users want or need while user feedback given on the results of application queries may help to identify more precisely if the information provided by a given data set is relevant or not. In general terms, if the user considers that the query results are "good" then the data set may be considered relevant.

[1]http://dblp.rkbexplorer.com/
[2]http://dblp.l3s.de/d2r/
[3]http://dbpedia.org/
[4] http://www.geonames.org/
[5]http://pubmed.bio2rdf.org
[6]http://rae2001.rkbexplorer.com/

3. PRELIMINARY DEFINITIONS

In this section, we present some preliminary definitions and notations that we use throughout the paper.

3.1 SPARQL Query

The main parts of a SPARQL query are [10]: the *pattern matching* part, which is composed by features of pattern matching of graphs; the *solution modifiers*, which allow modifying the output of the pattern and the *output*, which specifies the result form of the query.

In this work, we are interested on SELECT SPARQL queries whose final form of the result is a table. The columns of the table correspond to the values obtained from the matching of the variables specified in the SELECT clause against the graphs considered when computing the answer and according to the pattern described in the WHERE clause.

As an example, let us consider the SPARQL query given in Figure 1 that extracts information from DBPedia about titles of The Beatles songs. We can identify in such a query: i) the Query Result Form *SELECT ?title*, ii) the Basic Graph Pattern (or BGP) contained within WHERE *{?bandWorkdbpprop:artist ?band.} {?bandWorkfoaf:name ?title.} FILTER (?band = dbpedia:The_Beatles)}* section that describes the patterns the resulting triples should all match.

Query 1. Return the titles of all The Beatles songs.

```
SELECT ?title WHERE
{
        { ?bandWorkdbpprop:artist ?band. }
        { ?bandWorkfoaf:name ?title . }
        FILTER (?band = dbpedia:The_Beatles)
}
```

Figure 1. SPARQL query example

3.2 User Feedback

Users may annotate SELECT SPARQL queries results to specify if a given tuple was expected in the answer table of a query q or not. We call feedback of a query q

$$uf(q) = \{(t_1, v_1), (t_2, v_2), \dots, (t_k, v_k)\} \quad (3.1)$$

the set of annotations over the results of q, such that an annotation may be defined by the pair: (t, v), where t is a tuple in the answer table of q and v is the feedback value, which can be one of the following terms [2]: i) true positive (tp): a given instance was expected in the answer; ii) false positive (fp): a certain instance was not expected in the answer and iii) false negative (fn): an expected instance was not retrieved.

Table 1. User Feedback of Query 1

Id	*Title*	*Feedback*
t_1	"Rock and Roll Music"	tp
t_2	"Sgt. Pepper's Lonely Hearts Club Band"	tp
t_3	"What Goes On"	tp
t_4	"Hello, Goodbye"	fp
t_5	"Hey Jude"	fn

As an example assume that the evaluation results of *Query 1* are displayed as shown in Table 1. The user examines such results and provides feedback specifying whether they meet the

requirements. For example, the feedback instance uf given below specifies that the tuple t_1 is a true positive, i.e., meets the user's expectations, while tuple t_4 is a false negative, i.e., the result was expected by the user and it was not returned.

$uf(Query\ 1) = \{$ ("Rock and Roll Music", tp),

("Sgt.Pepper's Lonely Hearts Club Band", tp),

("What Goes On", tp), ("Hello, Goodbye", fn), ("Hey Jude", fn)}

Given the feedback annotations, it is possible to obtain values for precision and recall of the results of a given query q. Specifically, $precision$ is defined as the ratio of the number of true positives to the sum of true positives and false positives of $uf(q)$. Similarly, $recall$ is the ratio of the number of true positives to the sum of true positives and false negatives of $uf(q)$. From these we may calculate $f_{measure}$ value, which is defined as the harmonic mean of the precision and recall. We will use these values for calculating the relevance analysis proposed in this paper.

$$P(q) = \frac{tp(q)}{tp(q) + fp(q)} \quad (3.2)$$

$$R(q) = \frac{tp(q)}{tp(q) + fn(q)} \quad (3.3)$$

$$F(q) = \frac{2 * P(q) * R(q)}{P(q) + R(q)} \quad (3.4)$$

4. PROPOSED APPROACH

In this work we propose an approach for recommending data sets for domain-specific linked data applications. The recommendation is based on a set of queries that reflect the user requirements and on the user feedback given on the results of such queries. We assume that during application design, an initial set Q of significant application queries is defined and a data set ds is chosen to be the initial one. In addition, queries from Q are executed over ds and the user gives feedback on the corresponding result. It is important to note that the proposed approach is domain-independent, i.e., it will work with data sets of any domain.

In this section we present the proposed approach, which consists of two main phases. The first one searches the Web of Data to discover new data sets, whereas the second phase ranks the discovered data sets to identify the most relevant ones.

4.1 Data Sets Filtering

The increasing number of data sets becoming available on the Web of Data arises the need for mechanisms that help filtering data sets according to some criteria. In our approach, we are interested in filtering data sets according to application queries, i.e. we are interested in finding data sets that may help improving the results of the most frequently queries posed to the system. The filtering process is performed by the algorithm presented in Algorithm 1 and described as follows.

The algorithm receives a set of application queries as input and gives a list of candidate relevant data sets as output (a set of SPARQL endpoints). The first step of the algorithm consists in using the function *ExtractRelevantResources* in order to identify the set of relevant query resources, which will guide the search for candidate data sets. A query resource may be a subject or object of a triple and consists of a URI. The *ExtractRelevantResources* function (Algorithm 2) receives as input the set Q and returns a list of the most frequent resources of Q. Specifically, the resources extraction consists in retrieving the BGP for each one of the queries q of Q and for each triple pattern (triplePattern) of a given BGP, their elements (subject, predicate and object) are visited in order to build a list of resources and their respective occurrence. At the end of this process, the top-k resources will be selected as the most relevant ones.

Algorithm DatasetsFiltering
Input Q: A set of queries
k: Number of relevant resources to be considered during the crawling
Output *DE:* A set of SPARQL endpoints of fetched datasets
Begin
1. $RR \leftarrow \boldsymbol{ExtractRelevantResources}(Q)$
2. $Seeds \leftarrow SelectResources(RR, k)$
3. $Predicates \leftarrow \{sameAs,\ seeAlso,\ equivalentClass\}$
4. $FetchedTriples \leftarrow ExecuteCrawling(Seeds, Predicates)$
5. $ProvenanceTriples \leftarrow ExtractProvenance(FetchedTriples)$
6. $DE \leftarrow \emptyset$
7. **For each** $p \in ProvenanceTriples$ **do**
8. $DE \leftarrow DE \cup RetrieveSparqlEndpoint(p)$
9. **End for**
10. **Return** DE
End

Algorithm 1. Data sets Filtering

Algorithm ExtractRelevantResources

Input Q: A set of queries
Output *RR:* A sorted list by frequency of query resources
Begin
1. $FrequencyList \leftarrow \emptyset$
2. **For each** $q \in Q$ **do**
3. $BGP \leftarrow ExtractBGP(q)$
4. **For each** $triplePattern \in BGP$ **do**
5. $Resources \leftarrow VisitTriplePattern(triplePattern)$
6. $FrequencyList \leftarrow FrequencyList \cup Resources$
7. **End for**
8. **End for**
9. $RR \leftarrow DecreasingElementsList(FrequencyList)$
10. **Return** RR
End

Algorithm 2. Extraction of Relevant Resources

Once the most relevant resources were identified, the next step consists of crawling the Web of Data in order to find the set of candidate data sets. The crawling process considers as seeds the top-k resources of the list RR (Relevant Resources), which is composed by URIs that represent relevant resources according to Q. A set of predicates (*rdfs:seeAlso*, *owl:sameAs* and *owl:equivalentClass*) is used during the crawling to obtain new resources that are similar to the ones of seeds. At the end of the crawling, a list of new resources is obtained. The next step of the filtering process consists in building the list of relevant candidate sets. For this, it is extracted the provenance of the triples obtained during the crawling process. Using the provenance URI of each triple stored, it is used the function *RetrieveSparqlEndpoint* which extracts the data sets from which the resources of the list were obtained.

4.2 Data Sets Relevance Analysis

Our proposal for evaluating the relevance of a data set considers as input a set $Q = \{q_1, \dots, q_n\}$ of SPARQL queries and a set of user feedback annotations $UF = \{uf(q_1), \dots, uf(q_n)\}$ given over the results of such queries. Queries from Q reflect the main application data requirements, while the set of feedback annotations helps to identify the information that users really want or need.

In order to refine the relevance analysis, for each query q is assigned a weight value. The $weight$ of a query q, denoted by $w(q)$, helps to identify the most important queries for the application. Consider two queries q_1 and q_2, with weights $w(q_1)$ and$w(q_2)$, respectively, if $w(q_1) > w(q_2)$ then query q_1 is more important than q_2. In our approach, the weight $w(q)$is defined by the execution frequency of q.

In this context, given a set of SPARQL queries $Q = \{q_1, \dots, q_n\}$, a set of weights $W = \{w(q_1), \dots w(q_n)\}$, a set of feedback instances $UF = \{uf(q_1), \dots, uf(q_n)\}$ and a candidate data set d, the relevance calculus of the data set may be defined as follows:

$$R(d) = \frac{\sum_{i=1}^{|Q|} B(d, q_i) * w(q_i)}{\sum_{i=1}^{|Q|} w(q_i)} \qquad (4.1)$$

Where $Benefit\ B(d, q_i)$ determines the level of influence of a data set d for a given query q_i. Benefit $B(d, q)$ may be calculated as defined below:

$$B(d, q) = \frac{F'}{F} \qquad (3.5)$$

Where F is the value of $f_{measure}$ obtained from $uf(q)$, such that $uf(q)$ is the set of annotations over the results of q when q is evaluated over a set of data sets that does not include the candidate data set d; and F' is the value of $f_{measure}$ obtained from $uf'(q)$, where $uf'(q)$ is the set of annotations over the results of q when q is evaluated over a set of data sets that includes the candidate data set d.

In a general way, we can say that a data set d has a good influence over a query q if richer query answers are obtained when q is evaluated over a set of data sets, which includes the data set d. Considering the user feedback previously defined, d has a good influence over q if $tp(q)$ augments or if $fp(q)$ reduces. The algorithm for computing the relevance of a given data set is presented in Algorithm 3 and described below.

Initially, for each one of the queries q of Q, the following steps are performed. Using the function *CalculateF_measure* the algorithm computes the values of precision, recall and f-measure based on annotations provided by the user feedback (line 2). The next step consists of rewriting the query q in order to allow its execution over d (lines 3-4). The rewriting of query q becomes necessary because (i) the query q was initially submitted over a data set different from d and (ii) the calculus of the $benefit$ of d depends on the feedback given over the results of q when q is evaluated considering a set of data sets that includes d. It is important to note that the query rewriting process is out of the scope of this work. More details about this task can be found in other works proposed in the literature [1,4,6,8].

Once the rewritten query q' is obtained and executed over d, the function *InferFeedback* (Algorithm 4) is used to infer feedback annotations for the results of the rewritten query. This is done in order to avoid that the user has to give a new feedback every time that a new data set is being evaluated. The *InferFeedback* function infers feedback annotations for the results of the rewritten query q'. Such inference is based on the feedback annotations previously given over the results of q, in such a way that when a tuple t' in the answer table of q' is equivalent to a tuple t in the answer table of q, we have two cases: (i) the feedback value of t is true positive or false positive: in this case, the new feedback value of t' is the same of t (true positive or false positive, respectively) and (ii) the feedback value of t is false negative: in this case the feedback value of t' receives false negative. However, if there is no tuple t' that is equivalent to t and the feedback value of t is false negative then a new annotation (t, fn) should be included in the set of inferred feedback annotations. Next, new values of precision, recall and f-measure are calculated (line 6) and a relevance value (line 7) is computed based on the values of F, F' and the $weight$ of q. After doing this for each query q, the relevance value of the data set d is calculated and its value (line 11) is returned.

The *RelevanceAnalysis* algorithm has to be executed for each one of the data sets identified during the filtering phase. As a result, a ranking list is created and the top-k most candidate relevant data sets will be recommended to be included in the application.

Algorithm RelevanceAnalysis
Inputs Q: A set of queries
UF: A set of feedback instances
W: A set of weights
d: a candidate data set
Output *Relevance*: A new relevance value
Begin
1. ***For each*** $q \in Q$ ***do***
2. $UF \leftarrow CalculateF\_measure(uf(q))$
3. $q' \leftarrow RewriteQuery(q, d)$
4. $Result \leftarrow ExecuteQuery(q')$
5. $uf(q') \leftarrow \boldsymbol{InferFeedback}(uf(q), Result)$
6. $UF' \leftarrow CalculateF\_measure(uf(q'))$
7. $Relevance \leftarrow Relevance + (F'/F * w(q))$
8. $Sum \leftarrow Sum + w(q)$
9. ***End for***
10. $Relevance \leftarrow Relevance\ /\ Sum$
11. ***Return*** *Relevance*
End

Algorithm 3. Relevance Analysis

Algorithm InferFeedback
Inputs *uf(q)*: A user feedback of query q
Result(q'): A set of tuples obtained over execution of the query q'
Output *uf'(q')*: A inferred feedback of query q'
Begin

1. ***For each*** $annotationa \in uf$ ***do***
2. ***For each*** $result\ r \in Result$ ***do***
3. ***If*** $(t(a) = r)$ ***then***
4. ***If*** $(v(a) =$ *'True Positive'* ***OR*** $v(a) =$ *'False Negative'*) ***then***
5. $v'(a) \leftarrow$ *'True Positive'*
6. ***Else*** $v'(a) \leftarrow$ *'False Positive'*
7. $f_{inferred} \leftarrow f_{inferred} + \{(t(a), v'(a))\}$
8. ***End If***
9. ***Else If*** $(v(a) =$ *'FalseNegative'*) ***then***
10. $v'(a) \leftarrow$ *'False Negative'*
11. $f_{inferred} \leftarrow f_{inferred} + \{(t(a), v'(a))\}$
12. ***End If***
13. ***End If***
14. ***End for***
15. ***End for***
16. ***Return*** $f_{inferred}$
End

Algorithm 4. Feedback Inference

4.3 An example

To illustrate the data sets recommendation process consider the following example. Suppose an application that plans to offer information about *The Beatles*. Let $Q = \{q_1, q_2, q_3\}$, presented in Figure 2, be the set of initial queries defined during the

application design and DBpedia the data set chosen to be the base data source.

q1: Return the titles of all The Beatles songs
SELECT ?title WHERE{ *{ ?bandWorkdbpprop:artist ?band. }* *{ ?bandWorkfoaf:name ?title . }* *FILTER (?band = dbpedia:The_Beatles)* *}*
q2: Return the member names and his spouse name
SELECT ?memberName ?memberSpouseName WHERE{ *{ dbpedia:The_Beatlesdbpedia-owl:bandMember ?member . }* *{ ?member foaf:name ?memberName . }* *{ ?memberSpousedbpedia-owl:spouse ?member . }* *{ ?memberSpousefoaf:name ?memberSpouseName . }* *}*
q3: Return the websites of the band The Beatles
SELECT ?memberName ?memberPage WHERE{ *{dbpedia:The_Beatlesdbpedia-owl:bandMember ?member . }* *{ ?member foaf:name ?memberName . }* *{ ?member foaf:page ?memberPage . }* *}*

Figure 2. Example queries

Initially, queries from Q are executed over DBpedia and the corresponding results are annotated by one of the application users. Figure 3 presents some of the provided feedback annotations.

$uf(q1)$ = { ("Rock and Roll Music", tp), ("Sgt.Pepper's Lonely Hearts Club Band", tp), ("What Goes On", tp),("Hello, Goodbye", fn),("Hey Jude", fn)}
$uf(q2)$ = { ("John Lennon", "Yoko Ono", tp), ("Ringo Starr", "Barbara Bach", fn), ("Ringo Starr", "Maureen Cox", fp), ("Paul McCartney", "Linda McCartney", tp) }
$uf(q3)$ = { ("Ringo Starr", http://dbpedia.org/resource/Ringo_Starr , tp), ("MBE", http://dbpedia.org/resource/PaulMcCartney, fp), ("McCartney", http://dbpedia.org/resource/McCartney, fn)}

Figure 3. Feedback annotations for tuples obtained from DBpedia

Next, based on queries from Q is executed the data sets filtering phase. Initially, the most relevant resources are extracted from the BGP of queries q_1, q_2 and q_3. Once the most relevant resources were identified, the crawling process starts with the goal of finding new data sets capable of providing resources similar to the ones obtained from the BGP. Table 2 presents a list of similar resources identified during the crawling. The provenance of these resources is analyzed and then the data sets BBCMusic[7] and MusicBrainz[8] are identified as the candidate relevant data sets.

The next step consists of analyzing the relevance of the candidate data sets with respect to the user requirements gathered through the user feedback. Consider, for example, the data set BBCMusic. Initially, values for precision, recall and f-measure are calculated for query q_1 considering the feedback annotations given on the answer obtained from DBpedia (Figure 3). As a result, the value $f_{measure}\ (q_1) = 0{,}7$ is obtained. Next, query q_1 is rewritten into a query q_1', which is executed over the data set BBCMusic. Then, tuples in the corresponding answer are annotated according to the inferred feedback (Figure 4) and new values for precision, recall and f-measure are calculated for query q_1'. As a result, the values $f'_{measure}\ (q_1) = 0{,}8$ and $B(\mathrm{BBC}, q_1) = 1{,}14$ are obtained. Considering such values and the weight of q_1 then it is calculated the relevance of BBCMusic with respect to q_1.

Table 2. List of resources identified during crawling the Web of Data

http://dbpedia.org/resource/The_Beatles
http://en.wikipedia.org/wiki/Ringo_Starr
http://dbpedia.org/resource/John_Lennon
http://dbpedia.org/resource/Ringo_Starr
http://en.wikipedia.org/wiki/George_Harrison
http://dbpedia.org/resource/Paul_McCartney

$uf(q1')$ = { ("Rock and Roll Music", tp), ("What Goes On", tp), ("Hello, Goodbye", fn),("Hey Jude", tp)}
$uf(q2')$ = { ("John Lennon", "Yoko Ono," tp), ("Ringo Starr", "Barbara Bach," tp), ("Paul McCartney", "Linda McCartney" , tp)}
$uf(q3')$ = { ("Ringo Starr", http://dbpedia.org/resource/Ringo_Starr ,tp), ("McCartney", http://dbpedia.org/resource/McCartney, fn)}

Figure 4. Inferred feedback annotations for tuples obtained from BBCMusic

The same procedure is performed for queries q_2 and q_3. Finally, based on relevance values of BBCMusic with respect toq_1, q_2 and q_3, it is calculated its relevance with respect to Q. The final relevance value is $R(BBCMusic) = 1{,}08$. Table 3 summarizes the values of f-measure, f'-measure and benefit used in this calculus.

Table 3. Benefit values for the data set BBCMusic

$f_{measure}\ (q_1) = 0{,}7$	$f'_{measure}\ (q_1) = 0{,}8$	$B(BBC, q_1) = 1{,}14$
$f_{measure}\ (q_2) = 0{,}67$	$f'_{measure}\ (q_2) = 0{,}75$	$B(BBC, q_2) = 1{,}12$
$f_{measure}\ (q_3) = 0{,}85$	$f'_{measure}\ (q_3) = 0{,}9$	$B(BBC, q_3) = 1{,}05$

In our approach the relevance value $R(d) \in [1, \infty)$. When $0 < R \leq 1$, we can say that the data set d has low or no influence over the application queries, i.e. data set d will be less relevant. However, when R increases, the influence of the data set over the application queries also increases. Then the higher the value of R, the better will be the data set for the application considering both the application queries and the user feedback.

The relevance analysis is also performed for the data set MusicBrainz. Then the user may choose which data set should be included in the application.

[7] www.bbc.co.uk/music
[8] musicbrainz.org/

5. IMPLEMENTATION ISSUES AND EXPERIMENTS

To validate our approach, a prototype has been developed and some experiments on the domain of bibliographic data were performed. Specifically, we considered that users were interested on information about Computer Science bibliography. Our scenario was composed of five SPARQL queries and DBLP RKBExplorer was chosen as the initial data set.

Initially, as the result of the data set filtering phase, three candidate data sets were returned: ACM[9], Citeseer[10] and Roma[11]. For each one of the candidate data sets, the relevance analysis process was performed. Figure 5 presents the benefit values for each data set with respect to each one of the five queries.

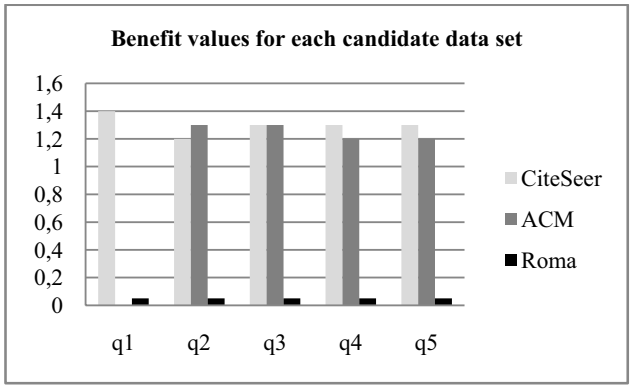

Figure 5. Benefit values for candidate data sets

It is important to note that for some queries, a data set may be more relevant than the others. For example, for query q_4 Citeseer has a benefit value better than ACM, while in query q_2 is the opposite. Based on these benefit values a relevance value was obtained for each one of the three data sets.

The Citeseer data set had the best relevance value ($R = 1{,}30$). In other words, this data set has contributed more than the others as it could contribute for the answering of the great majority of queries. In the ACM data set, we had a smaller contribution ($R = 1{,}14$) because only few queries could be affected by the inclusion of this source. The last one, the Roma data set had the worst result ($R = 0$) because almost no query would be affected by the addition of this source.

Considering these relevance values, we conclude that ACM and Citeseer data sets are good sources and then should be recommended as new data sets for the application because they may improve the quality of queries results. However, the Roma data set is not good. Its relevance value is low, i.e., it did not contribute to improve the query results.

We can also observe that the proposed approach selects a data set as a relevant one based on its relevance for the application instead of the quality of the data. In this setting, it is possible to avoid the selection of those data sets with bad quality. However, it is not possible to assure that the data sets with good data quality will be selected.

During our experiments, we faced some difficulties regarding the availability of data sets because some of them do not have an active and online SPARQL endpoint. Therefore, if the endpoint is not available then it is not possible to execute queries and consequently the data set couldn't be evaluated. Because of this, several good data sets couldn't be recommended. Other difficulty regards the blocking of the crawler access. In our approach, crawlers are used to obtain relevant resources based on the application queries in order to help to find out new candidate data sets. During our experiments, we found out that many data sets block the access of these crawlers, making it impossible to access its data.

Moreover, it should be noted that the number of queries is relevant to the proposed approach, however the quality of the user feedback is even more important. As a consequence, there will be situations where a small number of queries with feedback annotations of good quality will give better results than a larger set of queries with inconsistent feedback annotations, for example.

6. RELATED WORK

In this section we briefly present some of the research literature related to our work. The work presented in [9], for example, has the goal of identifying relevant sources for data linking. They propose an approach, which utilizes keyword-based search to find initial candidate sources for data linking, and ontology matching technologies as a way to assess the relevance of these candidates. Their approach has two main steps: (i) the searching for potentially entities in external data sources and (ii) the filtering of these sources using ontology matching techniques to filter out the irrelevant ones. They also apply a similarity measure between classes of the different sources in order to filter out the ones with low scores. One drawback of this proposal is that, in the filtering stage, only classes with stronger degree of semantic similarity are confirmed. In other words, many relevant classes may be filtered out because they are not considered as exact matches. Our work differs from this one in the sense that their approach focus on finding relevant data sets for linking instead of querying.

In the paper [12] the authors propose to find "dirty" sources using functional dependencies with probabilities (*p*FD) in the context of *pay-as-you-go* data integration systems. During the addition of a new data source, it is possible to decide if the source is good enough for the system based on the quality of the functional dependencies. It is important to mention that this approach just considers the relational model; in addition it also supposes the presence of a mediated schema.

In the work [11] is presented an approach to guide the addition of new sources in keyword search-based data integration systems. This process builds a search graph from the sources and its relationships. The search is performed over the graph and the results are returned in a top-k view with the most relevant answers to the user. The graph maintenance is made incrementally through user feedback and when new data sources are discovered the graph is realigned.

The work proposed in [7] uses the user feedback to rank mappings in *pay-as-you-go* systems. The approach uses the concept of VPI (value of perfect information) as a metric to rank. VPI provides a means of estimating the benefit to the *pay-as-you-go* system in such a way that it is possible to evaluate the correctness of a candidate matching based on the user feedback. This concept is based on the utility function that quantifies the quality of query's results. Similarly to VPI, in our approach we generate a relevance

[9]http://acm.rkbexplorer.com/
[10]http://citeseer.rkbexplorer.com/
[11]http://roma.rkbexplorer.com/

value. However, we are interested in rank data sets instead of ranking mappings.

7. CONCLUSION

Given the high heterogeneity and dynamicity of the data sets available on the Web of Data, it is possible to have data sets of poor quality, i.e., the data can be incorrect, incomplete or outdated. Therefore, having solutions that may help to identify good data sets to be used as input for a given application becomes crucial. In this paper, we presented an approach to recommend new data sets for domain-specific linked data applications. Among the main distinguishing issues of our approach, we highlight the use of application queries and feedback given by users, where application queries help to filter the Web of Data in order to find candidate relevant data sets, while the user feedback helps to analyze the relevance of such candidates. Considering both criteria, it is possible to find out data sets that really meet the user requirements. It is important to note that the proposed approach may be used in any application that needs to add new data sources regardless of the data model in which the application was designed.

To validate our approach, it was developed a prototype that performs both data sets filtering and data sets relevance analysis. The prototype also allows users to provide feedback through a GUI interface. Some experiments were performed to validate the behavior of the relevance value calculated according to the approach proposed in this paper.

As future work, we would highlight some directions:

(i) The improvement of techniques for the data sets filtering phase: we intend to adopt new data sets filtering techniques in order to minimize some of the difficulties previously described. For example, candidate data sets may be identified through the use of semantic indexes or a repository that stores information about the data sets may also be built. We will also investigate the use of VoiD descriptions in order to help or improve relevant data sets identification.

(ii) The improvement of the user feedback gathering: currently, we assume that a single user gives the feedback and if there are two annotations for the same tuple just the last one is considered the latest. We intend to apply more efficient techniques for user feedback gathering as well as for managing the user feedback evolution.

(iii) The improvement of the feedback inference algorithm: considering that feedback inference was not the main focus of our work, we applied just a simple technique for obtaining new feedback from the one already given by the user. We also intend to improve the quality of the inferred feedback, and thus improve the quality of the data set recommendation.

(iv) Case study on government data integration: we intend to build a case study considering open data from the Brazilian government. We plan to build a service to provide support for the development of applications that integrates data sets about the Brazilian government available on the Web, through the recommendation of relevant data sets.

8. REFERENCES

[1] Adjiman, P., Goasdoué, F., and Rousset, M. 2007. Somerdfs in the semantic web. *Journal on Data Semantics VIII*, Springer-Verlag, Berlin, Heidelberg, 2007, 158–181.

[2] Belhajjame, K., Paton, N. W., Embury, S. M., Fernandes, A. A. A., Hedeler C. 2010. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *Proceedings of the 13th International Conference on Extending Database Technology*, Lausanne, Switzerland, 2010, 573-584.

[3] Bizer, C., Heath, T., Berners-Lee, T. 2009. Linked data - the story so far. In *Proceedings of the International Journal on Semantic Web and Information Systems*, 5(3), 1-22.

[4] Calvanese, D., Giacomo, G. D., Lembo, D., Lenzerini, M., andRosati, R. 2004. What to ask to a peer: Ontology-based query reformulation. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning* (KR 2004), 469–478.

[5] Das Sarma, A., Dong, X. L., Halevy, A. 2011. Data Integration with dependent sources. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT 2011, New York, NY, USA.

[6] Fernandes, D. Y. S. 2009. Using Semantics to Enhance Query Reformulation in Dynamic Distributed Environments. PhD thesis, Federal University of Pernambuco.

[7] Jeffery S. R., Franklin M. J., Halevy A. Y. 2007. Soliciting User Feedback in a Dataspace System. Electrical Engineering and Computer Sciences University of California at Berkeley, 2007.

[8] Lopes, F. L. R., Sacramento, E. R., Loscio B. F. 2012. Using Heterogeneous Mappings for Rewriting SPARQL Queries. In *Proceedings of the 11th International Workshop on Web Semantics and Information Processing*, WebS 2012, Vienna, Austria, 2012. Accepted for publication.

[9] Nikolov, A., d'Aquin, M. 2011. Identifying Relevant Sources for Data Linking using a Semantic Web Index. In *Proceedings of the Linked Data on the Web*, LDOW 2011, Hyderabad, India.

[10] Pérez, J., Arenas, M., Gutierrez, C. 2009. Semantics and complexity of SPARQL. In *Proceedings of the ACM Transactions on Database Systems*, TODS 2009, Nova York, NY, USA, 34(3).

[11] Talukdar, P. P., Ives, Z. G., Pereira, F. 2010. Automatically incorporating new sources in keyword search-based data integration, In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD 2010, Indianapolis, Indiana, USA, 2010, 387-398.

[12] Wang, D. Z., Dong, X. L., Das Sarma, A., Franklin, M. J., Halevy, A. Y. 2009. Functional Dependency Generation and Applications in *Pay-as-you-go* Data Integration Systems. In *Proceedings of the 12th International Workshop on the Web and Databases*, WebDB 2009.