Question Answering on Interlinked Data

Saeedeh Shekarpour
Universität Leipzig, IFI/AKSW
shekarpour@informatik.uni-leipzig.de

Axel-Cyrille Ngonga Ngomo
Universität Leipzig, IFI/AKSW
ngonga@informatik.uni-leipzig.de

Sören Auer
Universität Leipzig, IFI/AKSW
auer@informatik.uni-leipzig.de

ABSTRACT

The Data Web contains a wealth of knowledge on a large number of domains. Question answering over interlinked data sources is challenging due to two inherent characteristics. First, different datasets employ heterogeneous schemas and each one may only contain a part of the answer for a certain question. Second, constructing a federated formal query across different datasets requires exploiting links between the different datasets on both the schema and instance levels. We present a question answering system, which transforms user supplied queries (i.e. natural language sentences or keywords) into conjunctive SPARQL queries over a set of interlinked data sources. The contribution of this paper is two-fold: Firstly, we introduce a novel approach for determining the most suitable resources for a user-supplied query from different datasets (disambiguation). We employ a hidden Markov model, whose parameters were bootstrapped with different distribution functions. Secondly, we present a novel method for constructing a federated formal queries using the disambiguated resources and leveraging the linking structure of the underlying datasets. This approach essentially relies on a combination of domain and range inference as well as a link traversal method for constructing a connected graph which ultimately renders a corresponding SPARQL query. The results of our evaluation with three life-science datasets and 25 benchmark queries demonstrate the effectiveness of our approach.

Categories and Subject Descriptors

I.2.7 [**Artificial intelligence**]:

General Terms

Algorithms, Human Factors

Keywords

Question answering, Hidden Markov Model, Linked Data, RDF, Disambiguation, SPARQL.

1. INTRODUCTION

There is a large and increasing quantity of structured data available on the Web. Traditional information retrieval approaches based on keyword search are user-friendly but cannot exploit the internal structure of data due to their bag-of-words semantic. For searching information on the Data Web we need similar user friendly approaches i.e. keyword-base interfaces, which leverage the internal structure of the data. Also, Question Answering is a specialized form of information retrieval. A Question Answering system attempts to extract correct answers to questions posed in natural language. Using the structure of data in retrieval process has two prominent advantages. Firstly, it approaches the information retrieval systems to question answering systems. Secondly, it enables us to easily integrate information from different datasets.

In this paper we present an approach for question answering over a set of interlinked data sources. We have to deal with two challenges: A first challenge is that information for answering a certain question can be spread among different datasets employing heterogeneous schemas. This makes the mapping of the input keywords to resources more challenging when compared to querying a single dataset. The second challenge is constructing a formal query from the matched resources across different datasets by exploiting links between the different datasets on the schema and instance levels.

In order to address these challenges, our approach resembles a horizontal search, where query segments derived from an input query are matched against all available datasets. We employ a Hidden Markov Model (HMM) to obtain the optimal input query segmentation and disambiguation of possible matches in a single step. We test different bootstrapping methods for the HMM parameters using various distributions (Normal, Zipf, Uniform) as well as an algorithm based on Hyperlink-Induced Topic Search (HITS). Our proposed functions for HMM parameters produce the best results for both segmentation and disambiguation. Then, we construct a formal query (expressed in SPARQL) using the disambiguated matches by traversing links in the underlying datasets. By taking links between the matched resources (including `owl:sameAs` links) into account we obtain the *minimum spanning graph* covering all matches in the different datasets.

As a test bed for evaluating our approach we used the Sider [1], Diseasome [8][2] and Drugbank [32][3] datasets published in RDF. Sider contains information about drugs and their side effects. Diseasome contains information about diseases and genes associated with these diseases. Drugbank is a comprehensive knowledge base containing information about drugs, drug target (i.e. protein) information, interactions and enzymes. As it can be seen in Figure 1 the classes representing drugs in Drugbank and Sider are linked using `owl:sameAs` and diseases from Diseasome are linked to drugs in Drugbank using `possible Drug` and `possible Disease target`. Diseases and side effects between Sider and Diseasome are linked using the `owl:sameAs` property. Note that in this figure the dotted arrows represent the properties between classes inside a dataset.

Our approach can answer queries with the following three characteristics:

- **Queries requiring fused information:** An example is the query: "side effects of drugs used for Tuberculosis". Tuberculosis is defined in Diseasome, drugs for curing Tuberculosis are described in Drugbank, while we find their side effects in Sider.
- **Queries targeting combined information:** An example depicted in Figure 2 is the query: "side effect and enzymes of drugs used for ASTHMA". Here the answer to that query can only be obtained by joining data from Sider (side effects) and Drugbank (enzymes, drugs).
- **Query requiring keyword expansion:** An example is the query "side effects of Valdecoxib". Here the drug Valdecoxib can not be found in Sider, however, its synonym Bextra is available via Sider.

To the best of our knowledge our approach is the first approach for answering questions on interlinked datasets by constructing a federated SPARQL query. Our main contributions can be summed up as follows:

- We extend the Hidden Markov Model approach for disambiguating resources from different datasets.
- We present a novel method for constructing formal queries using disambiguated resources and leveraging the interlinking structure of the underlying datasets.
- We developed a benchmark consisting of 25 queries for a testbed in the life-sciences. The evaluation of our implementation demonstrates its feasibility with an f-measure of 90%.

This paper is organized as follows: In the subsequent section, we present the problem at hand in more detail and some of the notations and concepts used in this work. Section 3 presents the proposed disambiguation method in detail along with the evaluation of the bootstrapping. In section 4, we then present the key steps of our algorithm for constructing a conjunctive query. Our evaluation results are presented in the section 5 while related work is reviewed in the section 6. We close with a discussion and future work.

2. PROBLEM AND PRELIMINARIES

In this section, we introduce some crucial notions employed throughout the paper and describe the main challenges that

[1]http://sideeffects.embl.de/
[2]http://diseasome.kobic.re.kr/
[3]http://www.drugbank.ca/

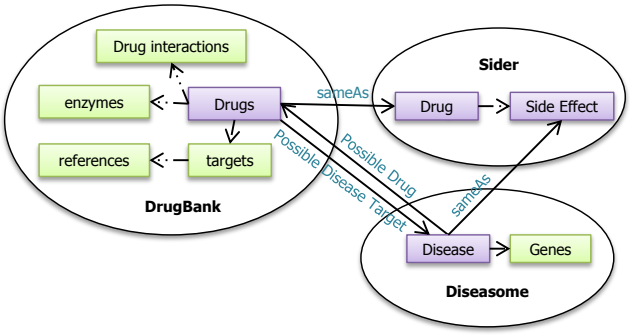

Figure 1: Schema interlinking for three datasets i.e. DrugBank, Sider, Diseasome.

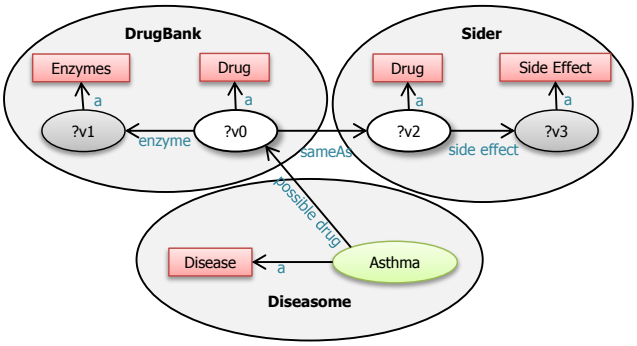

Figure 2: Resources from three different datasets are fused at the instance level in order to exploit information which are spread across diverse datasets.

arise when transforming user queries to formal, conjunctive queries on linked data.

An RDF knowledge base can be viewed as a directed, labeled graph $G_i = (V_i, E_i)$ where V_i is a set of nodes comprising all entities and literal property values, and E_i is a set of directed edges, i.e. the set of all properties. We define linked data in the context of this paper as a graph $G = (V = \bigcup V_i, E = \bigcup E_i)$ containing a set of RDF knowledge bases, which are linked to each other in the sense, that their sets of nodes overlap, i.e. that $V_i \cap V_j \neq \emptyset$.

In this work we focus on user-supplied queries in natural language, which we transform into an ordered sets of keywords by tokenizing, stop-word removal and lemmatization. Our input query thus is an n-tuple of keywords, i.e. $Q = (k_1, k_2, ..., k_n)$.

Challenge 1: Resource Disambiguation. In the first step, we aim to map the input keywords to a suitable set of entity identifiers, i.e. resources $R = \{r_1, r_2...r_m\}$. Note, that several adjacent keywords can be mapped to a single resource, i.e. $m \leq n$. In order to accomplish this task, the input keywords have to be grouped together to segments. For each segment, a suitable resource is then to be determined. The challenge here is to determine the right segment granu-

larity, so that the most suitable mapping to identifiers in the underlying knowledge base can be retrieved for constructing a conjunctive query answering the input query.

For example, the question *'What is the side effects of drugs used for Tuberculosis?'* is transformed to the 4-keyword tuple *(side, effect, drug, Tuberculosis)*. This tuple can be segmented into (*'side effect drug'*, *'Tuberculosis'*) or (*'side effect'*, *'drug'*, *'Tuberculosis'*). Note that the second segmentation is more likely to lead to a query that contains the results intended by the user. In addition to detecting the right segments for a given input query, we also have to map each of these segments to a suitable resource in the underlying knowledge base. This step is dubbed *entity disambiguation* and is of increasing importance since the size of knowledge bases and schemes heterogeneity on the Linked Data Web grows steadily. In this example, the segment *'drug'* is ambiguous when querying both Sider and Diseasome because it may refer to the resource `diseasome:Tuberculosis` describing the disease Tuberculosis or to the resource `sider:Tuberculosis` being the side effect caused by some drugs.

Challenge 2: Query Construction. Once the segmentation and disambiguation have been completed, adequate SPARQL queries have to be generated based on the detected resources. In order to generate a conjunctive query, a connected subgraph $G' = (V', E')$ of G called the **query graph** has to be determined. The intuition behind constructing such a query graph is that it has to fully cover the set of mapped resources $R = \{r_1, ..., r_m\}$ while comprising a minimal number of vertices and edges ($|V'| + |E'|$). In linked data, mapped resources r_i may belong to different graphs G_i; thus the query construction algorithm must be able to traverse the links between datasets at both schema and instance levels. With respect to the previous example, after applying disambiguation on the identified resources, we would obtain the following resources from different datasets: `sider:sideEffect`, `diseasome:possibleDrug`, `diseasome:1154`. The appropriate conjunctive query contains the following triple patterns:

```
1. diseasome:1154  diseasome:possibleDrug      ?v1 .
2. ?v1             owl:sameAs                  ?v2 .
3. ?v2             sider:sideEffect            ?v3 .
```

The second triple pattern bridges between the datasets Drugbank and Sider.

2.1 Resource Disambiguation

In this section, we present the formal notations for addressing the resource disambiguation challenge, aiming at mapping the n-tuple of keywords $Q = (k_1, k_2, ..., k_n)$ to the m-tuple of resources $R = (r_1, ..., r_m)$.

Definition 1 (Segment and Segmentation). *For a given query $Q = (k_1, k_2, ..., k_n)$, the segment $S_{(i,j)}$ is the sequence of keywords from start position i to end position j, i.e., $S_{(i,j)} = (k_i, k_{i+1}, ..., k_j)$. A query segmentation is an m-tuple of segments $SG(Q) = (S_{(0,i)}, S_{(i+1,j)}, ..., S_{(l,n)})$ with non-overlapping segments arranged in a continuous order, i.e. for two continuous segments S_x, S_{x+1} : $Start(S_{x+1}) = End(S_x) + 1$. The concatenation of segments belonging to a segmentation forms the corresponding input query Q.*

Valid Segments	Samples of Candidate Resources	
side effect	1. `sider:sideEffect`	2. `sider:side_effects`
drug	1. `drugbank:drugs`	2. `class:Offer`
	3. `sider:drugs`	4. `diseasome:possibledrug`
tuberculosis	1. `diseases:1154`	2. `side_effects:C0041296`

Table 1: Generated segments and samples of candidate resources for a given query.

Definition 2 (Resource Disambiguation). *Let the segmentation $SG' = (S^1_{(0,i)}, S^2_{(i+1,j)}, ..., S^x_{(l,n)})$ be the suitable segmentation for the given query Q. Each segment S^i of SG' is first mapped to a set of candidate resources $R_i = \{r_1, r_2...r_h\}$ from the underlying knowledge base. The aim of the disambiguation step is to detect an m-tuple of resources $(r_1, r_2, ..., r_m) \in R_1 \times R_2 \times ... \times R_m$ from the Cartesian product of the sets of candidate resources for which each r_i has two important properties: First, it is among the highest ranked candidates for the corresponding segment with respect to the similarity as well as popularity and second it shares a semantic relationship with other resources in the m-tuple. Semantic relationship refers to the existence of a path between resources.*

The disambiguated m-tuple is appropriate if a query graph [capable of answering the input query] can be constructed using all resources contained in that m-tuple. The order in which keywords appear in the original query is partially significant for mapping. However, once a mapping from keywords to resources is established the order of the resources does not affect the SPARQL query construction anymore. This is a fact that users will write strongly related keywords together, while the order of only loosely related keywords or keyword segments may vary. When considering the order of keywords, the number of segmentations for a query Q consisting of n keywords is $2^{(n-1)}$. However, not all these segmentations contain valid segments. A *valid segment* is a segment for which at least one matching resource can be found in the underlying knowledge base. Thus, the number of segmentations is reduced by excluding those containing invalid segments.

Algorithm 1 shows a naive approach for finding all valid segments when considering the order of keywords. It starts with the first keyword in the given query as first segment, then adds the next keyword to the current segment and checks whether this addition would render the new segment invalid. This process is repeated until we reach the end of the query. The input query is usually short. The number of keywords is mainly less than 6 [4]; therefore, this algorithm is not expensive. Table 1 shows the set of valid segments along with some samples of the candidate resources computed for the previous example using the naive algorithm. Note that 'side effect drug', 'side', 'effect' are not a valid segments.

2.2 Construction of Conjunctive Queries

The second challenge addressed by this paper tackles the problem of generating a federated conjunctive query leveraging the disambiguated resources i.e. $R = (r_1, ..., r_m)$. Herein, we consider conjunctive queries being conjunctions

[4] http://www.keyworddiscovery.com/keyword-stats.html?date=2012-08-01

```
Data: q: n-tuple of keywords, knowledge base
Result: SegmentSet: Set of segments
SegmentSet=new list of segments;
start=1;
while start <= n do
    i = start;
    while S_(start,i) is valid do
        SegmentSet.add(S_(start,i));
        i++;
    end
    start++;
end
```

Algorithm 1: Naive algorithm for determining all valid segments taking the order of keywords into account.

of SPARQL algebra triple patterns[5]. We leverage the disambiguated resources and implicit knowledge about them (i.e. types of resources, interlinked instances and schema as well as domain and range of resources with the type property) to form the triple patterns.

For instance, for the running query which asks for a list of resources (i.e. side effects) which have a specific characteristic in common (i.e. caused by drugs used for Tuberculosis'). Suppose the resources identified during the disambiguation process are: `sider:sideEffect`, `Diseasome:possibleDrug` as well as `Diseasome:1154`. Suitable triple patterns which are formed using the implicit knowledge are:

```
1. Diseasome:1154  Diseasome:possibleDrug      ?v1 .
2. ?v1             owl:sameAs                  ?v2 .
3. ?v2             sider:sideEffect            ?v3 .
```

The second triple pattern is formed based on interlinked data information. This triple connects the resources with the type `drug` in the dataset Drugbank to their equivalent resources with the type `drug` in the Sider dataset using `owl:sameAs` link. These triple patterns satisfy the information need expressed in the input query. Since most of common queries commonly lack of a quantifier, thus conjunctive queries to a large extend capture the user information need. A conjunctive query is called query graph and formally defined as follows.

DEFINITION 3 (QUERY GRAPH). *Let a set $R = \{r_1, ..., r_n\}$ of resources (from potentially different knowledge bases) be given. A query graph $QG_R = (V', E')$ is a directed, connected multi-graph such that $R \subseteq E' \cup V'$. Each edge $e \in E'$ is a resource that represents a property from the underlying knowledge bases. Two nodes n and $n' \in V'$ can be connected by e if n (resp. n') satisfies the domain (resp. range) restrictions of e. Each query graph built by these means corresponds to a set of triple patterns. i.e. $QG \equiv \{(n, e, n') | (n, n') \in V^2 \wedge e \in E\}$.*

3. RESOURCE DISAMBIGUATION USING HIDDEN MARKOV MODELS

In this section we describe how we use a HMM for the concurrent segmentation of queries and disambiguation of resources. First, we introduce the notation of HMM parameters and then we detail how we bootstrap the parameters of our HMM for solving the query segmentation and entity disambiguation problems.

Hidden Markov Models: Formally, a hidden Markov model (HMM) is a quintuple $\lambda = (X, Y, A, B, \pi)$ where:

- X is a finite set of states. In our case, X is a subset of the resources contained in the underlying graphs.
- Y denotes the set of observations. Herein, Y equals to the valid segments derived from the input n-tuple of keywords.
- $A : X \times X \rightarrow [0, 1]$ is the transition matrix of which each entry $a_{ij} =$ is the transition probability $Pr(S_j|S_i)$ from state i to state j;
- $B : X \times Y \rightarrow [0, 1]$ represents the emission matrix. Each entry $b_{ih} = Pr(h|S_i)$ is the probability of emitting the symbol h from state i;
- $\pi : X \rightarrow [0, 1]$ denotes the initial probability of states.

Commonly, estimating the hidden Markov model parameters is carried out by employing supervised learning. We rely on *bootstrapping*, a technique used to estimate an unknown probability distribution function. Specifically, we bootstrap[6] the parameters of our HMM by using string similarity metrics (i.e., *Levenshtein* and *Jaccard*) for the emission probability distribution and more importantly the topology of the graph for the transition probability. The results of the evaluation show that by using these bootstrapped parameters, we achieve a mean reciprocal rank (MRR) above 84%.

Constructing the State Space: A-priori, the state space should be populated with as many states as there are entities in the knowledge base. The number of states in X is thus potentially large given that X will contain all RDF resources contained in the graph G on which the search is to be carried out, i.e. $X = V \cup E$. For DBpedia, for example, X would contain more than 3 million states. To reduce the number of states, we exclude irrelevant states based on the following observations: (1) A relevant state is a state for which a valid segment can be observed (we described the recognition of valid segments in Section 2.1). (2) A valid segment is observed in a state if the probability of emitting that segment is higher than a given threshold θ. The probability of emitting a segment from a state is computed based on the similarity score which we describe in Section 3.1. Thus, we can prune the state space such that it contains solely the subset of the resources from the knowledge bases for which the emission probability is higher than θ. In addition to these states, we add an **unknown entity state** (UE) which represents all entities that were pruned. Based on this construction of state space, we are now able to detect likely segmentations and disambiguation of resources, the segmentation being the labels emitted by the elements of the most likely sequence of states. The disambiguated resources are the states determined as the most likely sequence of states.

Extension of State Space with reasoning: A further extension of the state space can be carried out by including resources inferred from lightweight `owl:sameAs` reasoning. We precomputed and added the triples inferred from the symmetry and transitivity property of the `owl:sameAs` rela-

[5]Throughout the paper, we use the standard notions of the RDF and SPARQL specifications, such as graph pattern, triple pattern and RDF graph.

[6]For the bootstrapping test, we used 11 sample queries from the QALD benchmark 2012 training dataset.

tion. Consequently, for extending the state space, for each state representing a resource `x` we just include states for all resources `y`, which are in an `owl:sameAs` relation with `x`.

3.1 Bootstrapping the Model Parameters

Our bootstrapping approach for the model parameters A and π is based on the HITS algorithm and semantic relations between resources in the knowledge base. The rationale is that the semantic relatedness of two resources can defined in terms of two parameters: the distance between the two resources and the popularity of each of the resources. The distance between two resources is the path length between those resources. The popularity of a resource is simply the connectivity degree of the resource with other resources available in the state space. We use the HITS algorithm for transforming these two values to hub and authority values (as detailed below). An analysis of the bootstrapping shows significant improvement of accuracy due to this transformation. In the following, we first introduce the *HITS* algorithm, since it is employed within the functions for computing the two HMM parameters A and π. Then, we discuss the distribution functions proposed for each parameter. Finally, we compare our bootstrapping method with other well-known distribution functions.

Hub and Authority of States. *Hyperlink-Induced Topic Search* (HITS) is a link analysis algorithm that was developed originally for ranking Web pages [13]. It assigns a hub and authority value to each Web page. The *hub value* estimates the value of links to other pages and the *authority value* estimates the value of the content on a page. Hub and authority values are mutually interdependent and computed in a series of iterations. In each iteration the authority value is updated to the sum of the hub scores of each referring page; and the hub value is updated to the sum of the authority scores of each referring page. After each iteration, hub and authority values are normalized. This normalization process causes these values to converge eventually.

Since RDF data forms a graph of linked entities, we employ a weighted version of the HITS algorithm in order to assign different popularity values to the states based on the distance between states. We compute the distance between states employing weighted edges. For each two states S_i and S_j in the state space, we add an edge if there is a path of maximum length k between the two corresponding resources. Note that we also take `property` resources into account when computing the path length.The weight of the edge between the states S_i and S_j is set to $w_{i,j} = k - pathLength(i,j)$, where $pathLength(i,j)$ is the length of the path between the corresponding resources. The authority of a state can now be computed by: $auth(S_j) = \sum_{S_i} w_{i,j} \times hub(S_i)$. The hub value of a state is given by $hub(S_j) = \sum_{S_i} w_{i,j} \times auth(S_i)$. These definitions of hub and authority for states are the foundation for computing the transition and initial probabilities in the HMM.

Transition Probability. To compute the transition probability between two states, we take both, the connectivity of the whole of space state as well as the weight of the edge between the two states, into account. The transition probability value decreases with increasing distance between states. For example, transitions between entities in the same triple have a higher probability than transitions between entities in triples connected through auxiliary intermediate entities. In addition to edges representing the shortest path between entities, there is an edge between each state and the *unknown entity (UE)* state. The transition probability of state S_j following state S_i is denoted as $a_{ij} = Pr(S_j|S_i)$. Note that the condition $\sum_{S_j} Pr(S_j|S_i) = 1$ holds.

The transition probability from the state S_i to UE is defined as:

$$a_{iUE} = Pr(UE|S_i) = 1 - hub(S_i)$$

Consequently, a good hub has a smaller probability of transition to *UE*. The transition probability from the state S_i to the state S_j is computed by:

$$a_{ij} = Pr(S_j|S_i) = \frac{auth(S_j)}{\sum_{\forall a_{ik}>0} auth(S_k)} \times hub(S_i)$$

Here, the probability from state S_i to the neighboring states are uniformly distributed based on the authority values. Consequently, states with higher authority values are more probable to be met.

Initial Probability. The initial probability $\pi(S_i)$ is the probability that the model assigns to the initial state S_i in the beginning. The initial probabilities fulfill the condition $\sum_{\forall S_i} \pi(S_i) = 1$. We denote states for which the first keyword is observable by $InitialStates$. The initial states are defined as follows:

$$\pi(S_i) = \frac{auth(S_i) + hub(S_i)}{\sum_{\forall S_j \in InitialStates} (auth(S_j) + hub(S_j))}$$

In fact, $\pi(S_i)$ of an initial state is uniformly distributed on both hub and authority values.

Emission Probability. Both the labels of states and the segments contain sets of words. For computing the emission probability of the state S_i and the emitted segment h, we compare the similarity of the label of state S_i with the segment h in two levels, namely string-similarity and set-similarity level:

- The *string-similarity level* measures the string similarity of each word in the segment with the most similar word in the label using the *Levenshtein distance*.
- The *set-similarity level* measures the difference between the label and the segment in terms of the number of words using the *Jaccard similarity*.

Our similarity score is a combination of these two metrics. Consider the segment $h = (k_i, k_{i+1}, ..., k_j)$ and the words from the label l divided into a set of keywords M and stopwords N, i.e. $l = M \cup N$. The total similarity score between keywords of a segment and a label is then computed as follows:

$$b_{ih} = Pr(h|S_i) = \frac{\sum_{t=i}^{j} \underset{m_i \in M}{\operatorname{argmax}}(\sigma(m_i, k_t))}{|M \cup h| + 0.1 * |N|}$$

This formula is essentially an extension of the *Jaccard similarity coefficient*. The difference is that we use the sum of

the string-similarity score of the intersections in the numerator instead of the cardinality of intersections. As in the Jaccard similarity, the denominator comprises the cardinality of the union of two sets (keywords and stopwords). The difference is that the number of stopwords is down-weighted by the factor 0.1 to reduce their influence since they do not convey much supplementary semantics.

Viterbi Algorithm for the K-best Set of Hidden States. The optimal path through the HMM for a given sequence (i.e. input query keywords) generates disambiguated resources which form a correct segmentation. The *Viterbi algorithm* or *Viterbi path* [29] is a dynamic programming approach for finding the optimal path through a HMM for a given input sequence. It discovers the most likely sequence of underlying hidden states that might have generated a given sequence of observations. This discovered path has the maximum joint emission and transition probability of the involved states. The sub-paths of this most likely path also have the maximum probability for the respective sub sequence of observations. The naive version of this algorithm just keeps track of the most likely path. We extended this algorithm using a tree data structure to store all possible paths generating the observed query keywords. Thus, our implementation can provide a ranked list of all paths generating the observation sequence with the corresponding probability. After running the Viterbi algorithm for our running example, the disambiguated resources are: {*sider:sideEffect, diseasome:possibleDrug, diseases:1154*} and consequently the detected segmentation is: {*side effect, drug, Tuberculosis*}.

3.2 Evaluation of Bootstrapping

We evaluated the accuracy of our approximation of the transition probability A (which is basically a kind of uniform distribution) in comparison with two other distribution functions, i.e., *Normal* and *Zipfian* distributions. Moreover, to measure the effectiveness of the *hub* and *authority* values, we ran the distribution functions with two different inputs, i.e. *distance* and *connectivity degree* values as well as *hub* and *authority* values. Note that for a given edge the source state is the one from which the edge originates and the sink state is the one where the edge ends. We ran the distribution functions separately with X being defined as the weighted sum of the normalized distance between two states and normalized connectivity degree of the sink state: $X_{ij} = \alpha \times distance_{(S_i - S_j)} + (1-\alpha) \times (1 - connectivityDegree_{S_j})$. Similarly, Y was defined as the weighted sum of the hub of the source state and the authority of the sink state: $Y = \alpha \times hub(S_i) + (1-\alpha) \times (1 - authority_{s_j})$. In addition, to measuring the effectiveness of *hub* and *authority*, we also measured a similar uniform function with the input parameters *distance* and *connectivity degree* defined as:

$$a_{ij} = \frac{distance(S_i - S_j)}{\sum\limits_{\forall S_k > 0} distance(S_i - S_k)} * connectivitydegree(S_i)$$

Given that the model at hand generates and scores a ranked list of possible tuples of resources, we compared the results obtained with the different distributions by looking at the *mean reciprocal rank* (MRR) [30] they achieve. For each query $q_i \in Q$ in the benchmark, we compare the rank r_i assigned by different algorithms with the correct tuple of resources and set $MRR(\mathcal{A}) = \frac{1}{|Q|} \sum\limits_{q_i} \frac{1}{r_i}$. Note that if the correct tuple of resources was not found, the reciprocal rank was assigned the value 0. We used 11 queries from QALD2-Benchmark 2012 training dataset for bootstrapping[7]. Figure 3 shows the MRR achieved by bootstrapping the transition probability of this model with 3 different distribution functions per query in 14 different settings. Figure 4 compares the average MRR for different functions employed for bootstrapping the transition probability per setting. Our results show clearly that the proposed function is superior to all other settings and achieves an MRR of approximately 81%. A comparison of the MRR achieved when using *hub* and *authority* with that obtained when using *distance* and *connectivity degree* reveals that using *hub* and *authority* leads to an 8% improvement on average. This difference is in Zipfian and Normal settings trivial, but very significant in the case of a uniform distribution. Essentially, *HITS* fairly assigns qualification values for the states based on the topology of the graph.

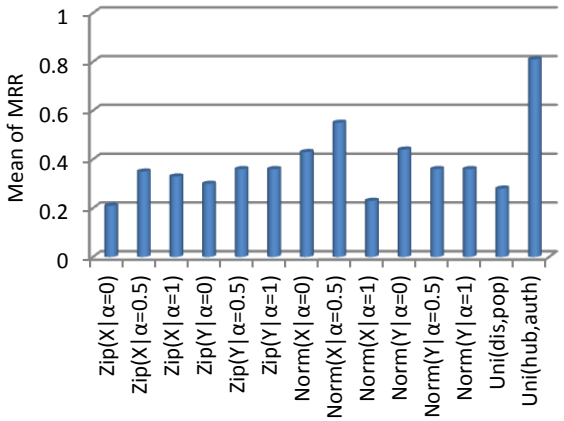

Figure 4: Comparison of different functions and settings for bootstrapping the transition probability. Uni stands for the uniform distribution, while Zip stands for the Zipfian and Norm for the normal distribution.

We bootstrapped the emission probability B with two distribution functions based on (1) Levenshtein similarity metric, (2) the proposed similarity metric as a combination of the Jaccard and Levenshtein measures. We observed the MRR achieved by bootstrapping the emission probability of this model employing those two similarity metrics per query in two settings (i.e. natural and reverse order of query keywords). The results show no difference in MRR between these two metrics in the natural order. However, in the reverse order the Levenshtein metric failed for 81% of the queries, while no failure was observed with the combination of Jaccard and Levenshtein. Hence, our combination is robust with regard to change of input keyword order. For bootstrapping the initial probability π, we compared the uniform distribution on both – hub and authority – values with a uniform distribution on the number of states for which the first keyword is observable. The result of this comparison shows a 5% improvement for the proposed function. Figure 5 shows the mean of MRR for different values of the threshold θ employed for prunning the state space. A high value of θ prevents inclusion of some relevant resources

[7] http://www.sc.cit-ec.uni-bielefeld.de/qald-2

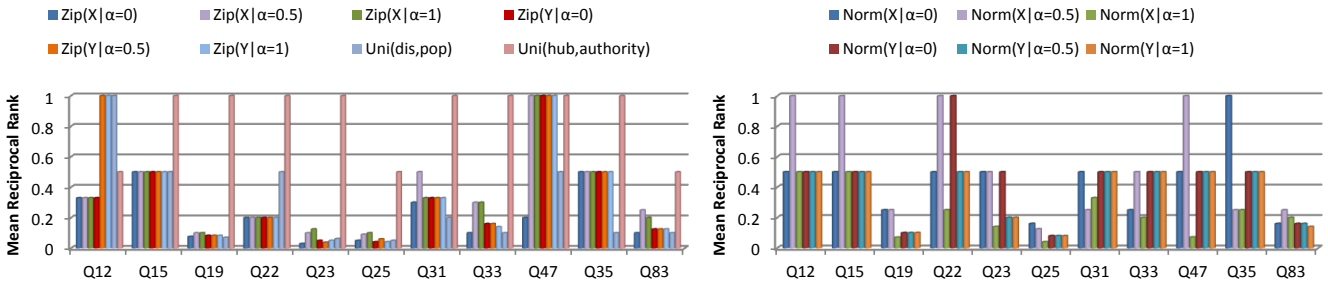

Figure 3: MRR of different distributions per query for bootstrapping the transition probability.

and a low value adds irrelevant resources. It can be observed that the optimal value of θ is in the range $[0.6, 0.7]$. Thus, we set θ to 0.7 in the rest of our experiments.

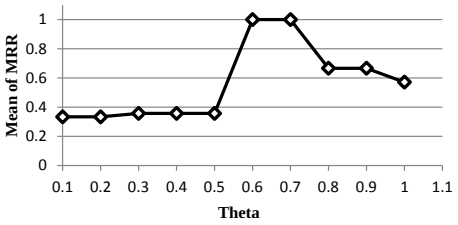

Figure 5: Mean MRR for different values of θ.

4. QUERY GRAPH CONSTRUCTION

The goal of query graph construction is generating a conjunctive query (i.e. SPARQL query) from a given set of resource identifiers i.e., $R = \{r_1, r_2, ...r_m\}$. The core of SPARQL queries are *basic graph patterns*, which can be viewed as a query graph QG. In this section, we first discuss the formal considerations underlying our query graph generation strategy and then describe our algorithm for generating the query graph. The output of this algorithm is a set of graph templates. Each graph template represents a comprehensive set of query graphs, which are isomorphic regarding edges. A query graph A is isomorphic regarding its edges to a query graph B, if A can be derived from B by changing the labels of edges.

4.1 Formal Considerations

A query graph QG consists of a conjunction of triple patterns denoted by (s_i, p_i, o_i). When the set of resource identifiers R is given, we aim to generate a query graph QG satisfying the *completeness* restriction, i.e., each r_i in R maps to at least one resource in a triple pattern contained in QG. For a given set of resources R, the probability of a generated query graph $\Pr(QG|R)$ being relevant for answering the information need depends on the probability of all corresponding triple patterns to be relevant. We assume that triple patterns are independent with regard to the relevance probability. Thus, we define the relevance probability for a QG as product of the relevance probabilities of the n containing triple patterns. We denote the triple patterns with $(s_i, p_i, o_i)_{i=1...n}$ and their relevance probability with $\Pr(s_i, p_i, o_i)$, thus rendering $\Pr(QG|R) = \prod_{i=1}^{n} \Pr(s_i, p_i, o_i)$. We aim at constructing QG with the highest relevance probability, i.e. $\arg\max \Pr(QG|R)$. There are two parameters that influence $\Pr(QG|R)$: (1) the number of triple patterns and (2) the number of free variables, i.e. variables in a triple pattern that are not bound to any input resource. Given that $\forall(s_i, p_i, o_i) : \Pr(s_i, p_i, o_i) \leq 1$, a low number of triple patterns increases the relevance probability of QG. Thus, our approach aims at generating small query graphs to maximize the relevance probability. Regarding the second parameter, more free variables increase the uncertainty and consequently cause a decrease in $\Pr(QG|R)$. As a result of these considerations, we devise an algorithm that minimizes the number of both the number of free variables and the number of triple patterns in QG. Note that is each triple pattern, the subject s_i (resp. object o_i) should be included in the domain (resp. range) of the predicate p_i or be a variable. Otherwise, we assume the relevance probability of the given triple pattern to be zero:

$$(s_i \notin domain(p_i)) \vee (o_i \notin range(p_i)) \Rightarrow \Pr(s_i, p_i, o_i) = 0.$$

Forward Chaining. One of the prerequisites of our approach is the inference of implicit knowledge on the types of resources as well as domain and range information of the properties. We define the *comprehensive type* (CT) of a resource r as the set of all super-classes of explicitly stated classes of r (i.e., those classes associated with r via the `rdf:type` property in the knowledge base). The comprehensive type of a resource can be easily computed using forward chaining on the `rdf:type` and `rdfs:subClassOf` statements in the knowledge base. We can apply the same approach to properties to obtain maximal knowledge on their domain and range. We call the extended domain and range of a property p *comprehensive domain* (CD_p) and *comprehensive range* (CR_p). We reduce the task of finding the *comprehensive properties* ($CP_{r-r'}$) which link two resources r and r' to finding properties p such that the comprehensive domain (resp. comprehensive range) of p intersects with the comprehensive type of r resp r' or vice-versa. We call the set OP_r (resp. IP_r) of all properties that can originate from (resp. end with) a resource r the set of outgoing (resp. incoming) properties of r.

4.2 Approach

To construct possible query graphs, we generate in a first step an *incomplete query graph* $IQG(R) = (V'', E'')$ such that the vertices V'' (resp. edges E'') are either equal or subset of the vertices (resp. edges) of the final query graph $V'' \subseteq V'$ (resp. $E'' \subseteq E'$). In fact, an incomplete query graph (IQG) contains a set of disjoint sub-graphs, i.e. there

is no vertex or edge in common between the sub-graphs: $IQG = \{g_i(v_i, e_i) | \forall g_i \neq g_j : v_i \cap v_j = \emptyset \wedge e_i \cap e_j = \emptyset\}$. An IQG connects a maximal number of the resources detected beforehand in all possible combinations.

The IQG is the input for the second step of our approach, which transforms the possibly incomplete query graphs into a set of final query graphs QG. Note that for the second step, we use an extension of the minimum spanning tree method that takes subgraphs (and not sets of nodes) as input and generates a minimal spanning graph as output. Since in the second step, the minimum spanning tree does not add any extra intermediate nodes (except nodes connected by `owl:sameAs` links), it eliminates both the need of keeping an index over the neighborhood of nodes and using exploration for finding paths between nodes.

Generation of IQGs. After identifying a corresponding set of resources $R = \{r_1, r_2, ...r_m\}$ for the input query, we can construct vertices V' and primary edges of the query graph $E'' \subseteq E'$ in an initial step. Each resource r is processed as follows: (1) If r is an instance, CT of this vertex is equivalent to $CT(r)$ and the label of this vertex is r. (2) If r is a class, CT of this vertex just contains r and the label of this vertex is a new variable.

After the generation of the vertices for all resources that are instances or classes, the remaining resources (i.e., the properties) generate an edge and zero (when connecting existing vertices), one (when connecting an existing with a new vertex) or two vertices. This step uses the sets of incoming and outgoing properties as computed by the forward chaining. For each resource r representing a property we proceed as follows:

- If there is a pair of vertices (v, v') such that r belongs to the intersection of the set of outgoing properties of v and the set of incoming properties of v' (i.e. $r \in OP_v \cap IP_{v'}$), we generate an edge between v and v' and label it with r. Note that in case several pairs (v, v') satisfy this condition, an IQG is generated for each pair.
- Else, if there is a vertex v fulfilling the condition $r \in OP_v$, then we generate a new vertex u with the CT_u being equal to CR_r and an edge labeled with the r between those vertices (v, u). Also, if the condition $r \in IP_v$ for v holds, a new vertex w is generated with CT_w being equal to CD_r as well as an edge between v and w labeled with r.
- If none of the above holds, two vertices are generated, one with CT equal to CD_r and another one with CT equal to CR_r. Also, an edge between these two vertices with label r is created.

This policy for generating vertices keeps the number of free variables at a minimum. Note that whenever a property is connected to a vertex, the associated CT of that vertex is updated to the intersection of the previous CT and CD_p (CR_p respectively) of the property. Also, there may be different options for inserting a property between vertices. In this case, we construct an individual IQG for each possible option. If the output of this step generates an IQG that contains one single graph, we can terminate as there is no need for further edges and nodes.

EXAMPLE 1. *We look at the query:* `What is the side effects of drugs used for Tuberculosis?`. *Assume the resource disambiguation process has identified the following resources:*

```
1. diseasome:possibleDrug      (type property)
   CD={diseasome:disease},     CR={drugbank:drugs}
2. diseasome:1154              (type instance)
   CT={diseasome:disease}
3. sider:sideEffect            (type property)
   CD={sider:drug},            CR={sider:sideeffect}
```

After running the IQGs generation, since we have only one resource with the type `class` *or* `instance`*, just one vertice is generated. Thereafter, since only the domain of* `possibleDrug` *intersects with the CT of the node* `1154`*, we generate: (1) a new vertex labeled* $?v0$ *with the CT being equal to* $CR =$`possibleDrug`*, and (2) an edge labeled* `possibleDrug` *from* `1154` *to* $?v0$*. Since, there is no matched node for the property* `sideEffect` *we generate: (1) a new vertex labeled* $?v1$ *with the CT being equal to* `sider:drug`*, (2) a new vertex labeled* $?v2$ *with the CT being equal to* `sider:sideeffect`*, (3) an edge labeled* `sideEffect` *from* $?v1$ *to* $?v2$*. Figure 6 shows the constructed IQG, which contains two disjoint graphs.*

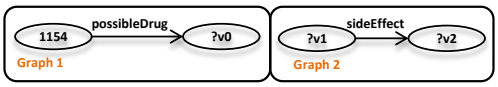

Figure 6: IQG for the Example 1.

Connecting Sub-graphs of an IQG. Since the query graph QG must be a connected graph, we need to connect the disjoint sub-graphs in each of the IQGs. The core idea of our algorithm utilizes the *Minimum Spanning Tree* (MST) approach, which builds a tree over a given graph connecting all the vertices. We use the idea behind Prim's algorithm [3], which starts with all vertices and subsequently incrementally includes edges. However, instead of connecting vertices we connect individual disjoint sub-graphs. Hence, we try to find a minimum set of edges (i.e., properties) to span a set of disjoint graphs so as to obtain a connected graph. Therewith, we can generate a query graph that spans all vertices while keeping the number of vertices and edges at a minimum. Since a single graph may have many different spanning trees, there may be several query graphs that correspond to each IQG. We generate all different spanning graphs because each one may represent a specific interpretation of the user query.

To connect two disjoint graphs we need to obtain edges that qualify for connecting a vertex in one graph with a suitable vertex in the other graph. We obtain these properties by computing the set of comprehensive properties CP (cf. Section 4.1) for each combination of two vertices from different sub-graphs. Note that if two vertices are from different datasets, we have to traverse `owl:sameAs` links to compute a comprehensive set of properties. This step is crucial for constructing a federated query over interlinked data. In order to do so, we first retrieve the direct properties between two vertices `?v0 ?p ?v1.` In case such properties exist, we add an edge between those two vertices to IQG. Then, we retrieve the properties connecting two vertices via an `owl:sameAs` link. To do that, we employ two graph patterns: (1) `?v0 owl:sameAs ?x. ?x ?p ?v1.` (2)

`?v0 ?p ?x. ?x owl:sameAs ?v1.` The resulting matches to each of these two patterns are added to the IQG. Finally, we obtain properties connecting vertices having `owl:sameAs` links according to the following pattern:
`?v0 owl:sameAs ?x. ?x ?p ?y. ?y owl:sameAs ?v1.` Also, matches for this pattern are added to the IQG.

For each connection discovered between a pair of vertices (v, v'), a different IQG is constructed by adding the found edge connecting those vertices to the original IQG. Note that the IQG resulting from this process contains less unconnected graphs than the input IQG. The time complexity in the worst case is $O(|v|^2)$ (with $|v|$ being the number of vertices).

EXAMPLE 2. *To connect two disjoint graphs i.e. Graph 1 and Graph 2 of the IQG shown in Example 1, we need to obtain edges that qualify for connecting either the vertex* `1154` *or* $?v0$ *to either vertex* $?v1$ *or* $?v2$ *in Graph 2. Forward chaining reveals the existence of two* `owl:sameAs` *connections between two vertices i.e. (1)* `1154` *and* $?v2$*, (2)* $?v0$ *and* $?v1$*. Therefore, we can construct the first query graph template by adding an edge between* `1154` *and* $?v2$ *and the second query graph template by adding an edge between* $?v0$ *and* $?v1$*. The two generated query graph templates are depicted in Figure 7.*

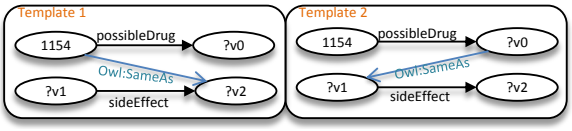

Figure 7: Generated query graph templates.

Our approach was implemented as a Java Web application which is publicly available at `http://sina-linkeddata.aksw.org`. The algorithm is knowledge-base-agnostic and can thus be easily used with other knowledge bases.

5. EVALUATION

Experimental Setup. The goal of our evaluation was to determine how well (1) our resource disambiguation and (2) our query construction approaches perform. To the best of our knowledge, no benchmark for federated queries over Linked Data has been created so far. Thus, we created a benchmark of 25 queries on the 3 interlinked datasets Drugbank, Sider and Diseasome for the purposes of our evaluation[8]. The benchmark was created by three independent SPARQL experts, which provided us with (1) a natural-language query and (2) the equivalent conjunctive SPARQL query. We selected these three datasets because they are a fragment of the well interlinked biomedical fraction of the Linked Open Data Cloud[9] and thus represent an ideal case for the future structure of Linked Data sources.

[8]The benchmark queries are available at `http://aksw.org/Projects/lodquery`

[9]For example, 859 `owl:sameAs` links exists between the 924 instances of drugs in Sider and the 4772 instances of drugs Drugbank

We measured the performance of our resource disambiguation approach using the *Mean Reciprocal Rank* (MRR). Moreover, we measured the accuracy of the query construction in terms of precision and recall. To compute the precision, we compared the results returned from the query construction method with the results of the reference query provided by the benchmark. The query construction is initiated with the top-1 tuple returned by the disambiguation approach. All experiments were carried out on a Windows 7 machine with an Intel Core2 Duo (2.66GHz) processor and 4GB of RAM. For testing the statistical significance of our results, we used a Wilcoxon signed ranked test with a significance level of 95%.

Results. The detailed results of our evaluation are shown in Figure 9. We ran our approach without and with OWL inferencing during the state space construction. When ran without inferencing, our approach was able to disambiguate 23 out of 25 (i.e. 92%) of the resources contained in the queries without mistakes. For Q9 (resp. Q25), the correct disambiguation was only ranked third (resp. fifth). In the other two cases (i.e. Q10 and Q12), our approach simply failed to retrieve the correct disambiguation. This was due to the path between `Doxil` and `Bextra` not being found for Q10 as well as the mapping from `disease` to `side effect` not being used in Q12. Overall, we achieve an MRR of 86.1% without inferencing. The MRR was 2% lower (not statistically significant) when including OWL inferencing due to the best resource disambiguation being ranked at the second position for three queries that were disambiguated correctly without inferencing (Q5, Q7 and Q20). This was simply due to the state space being larger and leading to higher transition probabilities for the selected resources. With respect to precision and recall achieved with and without reasoning, there were also no statistically significant differences between the two approaches. The approach without reasoning achieved a precision of 0.91 and a recall of 0.88 while using reasoning led to precision (resp. recall) values of 0.95 (resp. 0.90). Although performance was not (yet) the primary focus of our work, we want to provide evidence, that our approach can be used for real-time querying. Overall the pros and cons of using inferencing are clearly illustrated in the results of our experiments. On Q12, our approach is unable to construct a query without reasoning due to the missing equivalence between the terms `disease` and `side effect`. This equivalence is made available by the inference engine, thus making the construction of the SPARQL query possible. On the downside, adding supplementary information through inferencing alters the ranking of queries and can thus lead to poorer recall values as in the case of Q20.

Figure 8 shows the runtime average of disambiguation and query construction with and without inferencing during the state space construction for three runs. As it can be expected, inferencing increases the runtime, especially when the number of input keywords is high. Despite carrying out all computations on-the-fly, disambiguation and query construction terminate in reasonable time, especially for smaller number of keywords. After implementing further performance optimizations (e.g. indexing resource distances), we expect our implementation to terminate in less than 10s also for up to 5 keywords.

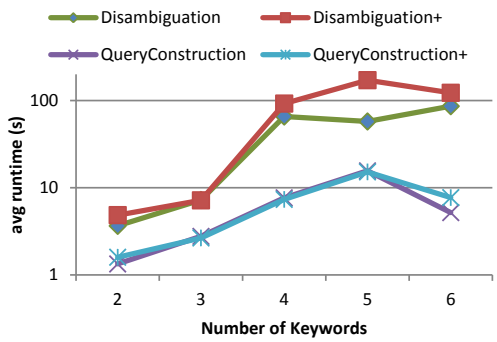

Figure 8: Average of Runtime of disambiguation and query construction with (+) and without reasoning in the disambiguation phase in logarithmic scale.

ID	Query	MRR	Pr	Re	MRR+	Pr+	Re+
1	Which are possible drugs against rickets?	1	1	1	1	1	1
2	Which are the drugs whose side effects are associated with the gene TRPM6?	1	1	1	1	1	1
3	Which diseases are associated with the gene FOXP2?	1	1	1	1	1	1
4	Which are targets of Hydroxocobalamin?	1	1	1	1	1	1
5	Which genes are associated with diseases whose possible drug targets Cubilin?	1	1	1	0.5	1	1
6	Which are possible drugs for diseases associated with the gene ALD?	1	1	1	1	1	1
7	Which are targets for possible drugs for diseases associated with the gene ALD?	1	1	1	0.5	1	1
8	Which are the side effects of Penicillin G?	1	1	1	1	1	1
9	Which drugs have hypertension and vomiting as side effects?	0.33	1	0.2	0.33	1	0.2
10	What are the common side effects of Doxil and Bextra?	0	0	0	0	0	0
11	Which diseases is Cetuximab used for?	1	1	1	1	1	1
12	What are the diseases caused by Valdecoxib?	0	0	0	1	1	1
13	What are the side effects of Valdecoxib?	1	1	1	1	1	1
14	What is the side effects of drugs used for Tuberculosis?	1	0.99	1	1	0.99	1
15	What are enzymes of drugs used for anemia?	1	1	1	1	1	1
16	What are diseases treated by tetracycline?	1	1	1	1	1	1
17	What are side effect and enzymes of drugs used for ASTHMA?	1	0.99	1	1	0.98	1
18	List references of drugs targeting Prothrombin!	1	1	1	1	1	1
19	What are drugs interacting with allopurinol?	1	1	1	1	1	1
20	What are associate genes of diseases treated with Cetuximab?	1	1	1	0.5	1	0.46
21	What is the food interaction of allopurinol?	1	1	1	1	1	1
22	Which drug does have fever as side effect?	1	1	1	1	1	1
23	What is the associated genes of breast cancer?	1	1	1	1	1	1
24	What is the target drug of Vidarabine?	1	1	1	1	1	1
25	Which drugs do target Multidrug resistance protein 1?	0.2	1	1	0.2	1	1

Figure 9: Accuracy results for the benchmark.

6. RELATED WORK

Several **information retrieval** and **question answering** approaches have been developed for the Semantic Web over the past years. Most of these approaches are adaptations of document retrieval approaches. For instance, *Swoogle* [5], *Watson* [4], *Sindice* [25] stick to the document-centric paradigm. Entity-centric approaches (e.g. *Sig.Ma* [24], *Falcons* [2], *SWSE* [11]) have recently emerged. However, the basis for all these services are keyword indexing and retrieval relying on the matching user keywords and indexed terms. Examples of question answering systems are *PowerAqua* [16] and *OntoNL* [12]. *PowerAqua* can automatically combine information from multiple knowledge bases at runtime. The input is a natural language query and the output is a list of relevant entities. PowerAqua lacks a deep linguistic analysis and can not handle complex queries. *Pythia* [27] is a question answering system that employs deep linguistic analysis. It can handle linguistically complex questions, but is highly dependent on a manually created lexicon. Therefore, it fails with datasets for which the lexicon was not designed. Pythia was recently used as kernel for *TBSL* [26], a more flexible question-answering system that combines Pythia's linguistic analysis and the *BOA framework* [7] for detecting properties to natural language patterns. Exploring schema from anchor points bound to input keywords is another approach discussed in [23]. Querying Linked datasets is addressed with the work mainly treat both the data and queries as bags of words [2, 31]. [10] presents a hybrid solution for querying linked datasets. It run the input query against one particular dataset regarding the structure of data, then for candidate answers, it finds and ranks the linked entities from other datasets . Our approach is a prior work as it queries all the datasets at hand and then according to the structure of the data, it makes a federated query. Furthermore, our approach is independent of any linguistic analysis and does not fail when the input query is an incomplete sentence.

Segmentation and disambiguation are inherent challenges of keyword-based search. Keyword queries are usually short and lead to significant keyword ambiguity [28]. Segmentation has been studied extensively in the natural language processing (NLP) literature e.g., [18]). NLP techniques for chunking such as part-of-speech tagging or name entity recognition cannot achieve high performance when applied to query segmentation. [17] addresses the segmentation problem as well as spelling correction and employs a dynamic programming algorithm based on a scoring function for segmentation and cleaning. An unsupervised approach to query segmentation in Web search is described in [21]. [33] is a supervised method based on Conditional Random Fields (CRF) whose parameters are learned from query logs. For detecting named entities, [9] uses query log data and Latent Dirichlet Allocation. In addition to query logs, various external resources such as Web pages, search result snippets, Wikipedia titles and a history of the user activities have been used [19, 22, 1, 20]. Still, the most common approach is using the context for disambiguation [15, 6, 14]. In this work, resource disambiguation is based on the structure of the knowledge at hand as well as semantic relations between the candidate resources mapped to the keywords of the input query.

7. DISCUSSION AND CONCLUSION

We presented a two-step approach for question answering from user-supplied queries over federated RDF data. A main assumption of this work is that some schema information is available for the underlying knowledge base and resources are typed according to the schema. Regarding the disambiguation, the superiority of our model is related to the transition probabilities. We achieved a fair balance between the qualification of states for transiting by reflecting the popularity and distance in the hub and authority values and setting a transition probability to the unknown entity state (depending on the hub value). This resulted in an accuracy of the generated answers of more than 90% for our test-bed with life-science datasets. This work represents a first step in a larger research agenda aiming to make the whole Data Web easily queryable. For scaling the implementation, a first avenue of improvements is related to the performance of the system, which can be improved by several orders of magnitued thorough including better indexing and precomputed forward-chaining.

8. REFERENCES

[1] D. J. Brenes, D. Gayo-Avello, and R. Garcia. On the fly query entity decomposition using snippets. *CoRR*, abs/1005.5516, 2010.

[2] G. Cheng and Y. Qu. Searching linked objects with falcons: Approach, implementation and evaluation. *Int. J. Semantic Web Inf. Syst.*, 5(3):49–70, 2009.

[3] D. R. Cheriton and R. E. Tarjan. Finding minimum spanning trees. *SIAM J. Comput.*, 1976.

[4] M. D'aquin, E. Motta, M. Sabou, S. Angeletou, L. Gridinoc, V. Lopez, and D. Guidi. Toward a new generation of semantic web applications. *Intelligent Systems, IEEE*, 23(3):20–28, 2008.

[5] L. Ding, T. W. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM*. ACM, 2004.

[6] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing Search in Context: the Concept Revisited. In *WWW*, 2001.

[7] D. Gerber and A.-C. Ngonga Ngomo. Extracting multilingual natural-language patterns for rdf predicates. In *Proceedings of EKAW*, 2012.

[8] K. Goh, M. Cusick, D. Valle, B. Childs, M. Vidal, and A. Barabási. Human diseasome: A complex network approach of human diseases. In *Abstract Book of the XXIII IUPAP International Conference on Statistical Physics*. 2007.

[9] J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. ACM, 2009.

[10] D. M. Herzig and T. Tran. Heterogeneous web data search using relevance-based on the fly data integration. pages 141–150. ACM, 2012.

[11] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and browsing linked data with swse: The semantic web search engine. *J. Web Sem.*, 9(4), 2011.

[12] A. Karanastasi, A. Zotos, and S. Christodoulakis. The OntoNL framework for natural language interface generation and a domain-specific application. In *First International DELOS Conference, Pisa, Italy*. 2007.

[13] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999.

[14] R. Kraft, C. C. Chang, F. Maghoul, and R. Kumar. Searching with context. In *WWW '06*. ACM, 2006.

[15] S. Lawrence. Context in web search. *IEEE Data Eng. Bull.*, 23(3):25–32, 2000.

[16] V. Lopez, F. M., M. E., and N. Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. In *Journal of Semantic Web*, In press.

[17] K. Q. Pu and X. Yu. Keyword query cleaning. *PVLDB*, 1(1):909–920, 2008.

[18] L. A. Ramshaw and M. P. Marcus. Text chunking using transformation-based learning. *CoRR*, 1995.

[19] K. M. Risvik, T. Mikolajewski, and P. Boros. Query segmentation for web search. 2003.

[20] A. Shepitsen, J. Gemmell, B. Mobasher, and R. Burke. Personalized recommendation in social tagging systems using hierarchical clustering. ACM, 2008.

[21] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. In *WWW*. ACM, 2008.

[22] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. ACM, 2008.

[23] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, 2009.

[24] G. Tummarello, R. Cyganiak, M. Catasta, S. Danielczyk, R. Delbru, and S. Decker. Sig.ma: Live views on the web of data. *J. Web Sem.*, 8(4):355–364, 2010.

[25] G. Tummarello, R. Delbru, and E. Oren. Sindice.com: weaving the open linked data. 2007.

[26] C. Unger, L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. ACM, 2012.

[27] C. Unger and P. Cimiano. Pythia: compositional meaning construction for ontology-based question answering on the semantic web. In *16th Int. Conf. on NLP and IS*, NLDB'11, pages 153–160, 2011.

[28] A. Uzuner, B. Katz, and D. Yuret. Word sense disambiguation for information retrieval. AAAI Press, 1999.

[29] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2), 1967.

[30] E. Vorhees. The trec-8 question answering track report. In *Proceedings of TREC-8*, 1999.

[31] H. Wang, Q. Liu, T. Penin, L. Fu, L. Z. 0007, T. Tran, Y. Yu, and Y. Pan. Semplore: A scalable ir approach to search the web of data. *J. Web Sem.*, 2009.

[32] D. S. Wishart, C. Knox, A. Guo, S. Shrivastava, M. Hassanali, P. Stothard, Z. Chang, and J. Woolsey. Drugbank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Research*, 34(Database-Issue), 2006.

[33] X. Yu and H. Shi. Query segmentation using conditional random fields. ACM, 2009.