Web Semantics: Science, Services and Agents on the World Wide Web 30 (2015) 39–51

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

SINA: Semantic interpretation of user queries for question answering on interlinked data

Saeedeh Shekarpour [a,b,*], Edgard Marx [b], Axel-Cyrille Ngonga Ngomo [b], Sören Auer [a,b,c]

[a] Department of Computer Science, IFI/AKSW, Universität Leipzig, Germany
[b] Department of Enterprise Information Systems (EIS), Institute for Applied Computer Science at University of Bonn, Germany
[c] Fraunhofer Institute for Intelligent Analysis and Information Systems, Bonn, Germany

ARTICLE INFO

Article history:
Received 16 July 2013
Received in revised form
23 May 2014
Accepted 18 June 2014
Available online 26 June 2014

Keywords:
Keyword search
Question answering
Hidden Markov model
SPARQL
RDF
Disambiguation

ABSTRACT

The architectural choices underlying Linked Data have led to a compendium of data sources which contain both duplicated and fragmented information on a large number of domains. One way to enable non-experts users to access this data compendium is to provide keyword search frameworks that can capitalize on the inherent characteristics of Linked Data. Developing such systems is challenging for three main reasons. First, resources across different datasets or even within the same dataset can be homonyms. Second, different datasets employ heterogeneous schemas and each one may only contain a part of the answer for a certain user query. Finally, constructing a federated formal query from keywords across different datasets requires exploiting links between the different datasets on both the schema and instance levels. We present SINA, a scalable keyword search system that can answer user queries by transforming user-supplied keywords or natural-languages queries into conjunctive SPARQL queries over a set of interlinked data sources. SINA uses a hidden Markov model to determine the most suitable resources for a user-supplied query from different datasets. Moreover, our framework is able to construct federated queries by using the disambiguated resources and leveraging the link structure underlying the datasets to query. We evaluate SINA over three different datasets. We can answer 25 queries from the QALD-1 correctly. Moreover, we perform as well as the best question answering system from the QALD-3 competition by answering 32 questions correctly while also being able to answer queries on distributed sources. We study the runtime of SINA in its mono-core and parallel implementations and draw preliminary conclusions on the scalability of keyword search on Linked Data.

1. Introduction

The principles underlying Linked Data have been applied worldwide to engender the Linked Open Data Cloud, a compendium of more than 300 datasets and more than 31 billions triples.[1] Within this compendium, millions of resources are described, partly over several datasets [1]. The current standard for accessing this wealth of data is the SPARQL query language. Yet, SPARQL is too complex to be used by non-expert users. Consequently, several search approaches have been developed over the last years to enable non-experts to access these datasets (e.g., [2–7]). While these approaches differ in their details (see Section 7), they can all be positioned on the following spectrum: on one end of the spectrum are simple keyword search systems that rely on traditional information retrieval approaches to retrieve resources that bear a label similar to the input of the user. We dub such approaches *data-semantics-unaware keyword search* as they do not take the semantics explicated by the data into consideration. The main advantage of such approaches is that they scale well as they can make use of the results of decades of research carried out in the field of information retrieval. On the other end of the spectrum, we find *question answering systems*, which assume a natural-language query as input and convert this query into a full-fledged SPARQL query. These systems rely on natural-language processing tools such as POS tagging and dependency parsers to detect the relations between the elements of the query. The detected relations are then mapped to SPARQL constructs.

The basic idea behind this work is to devise a *data-semantics-aware keyword search* approach, which stands in the middle of

* Corresponding author at: Department of Computer Science, IFI/AKSW, Universität Leipzig, Germany. Tel.: +49 17684419554.
E-mail address: sa.shekarpour@gmail.com (S. Shekarpour).

[1] See http://lod-cloud.net/state/ for more details.

http://dx.doi.org/10.1016/j.websem.2014.06.002
1570-8268/

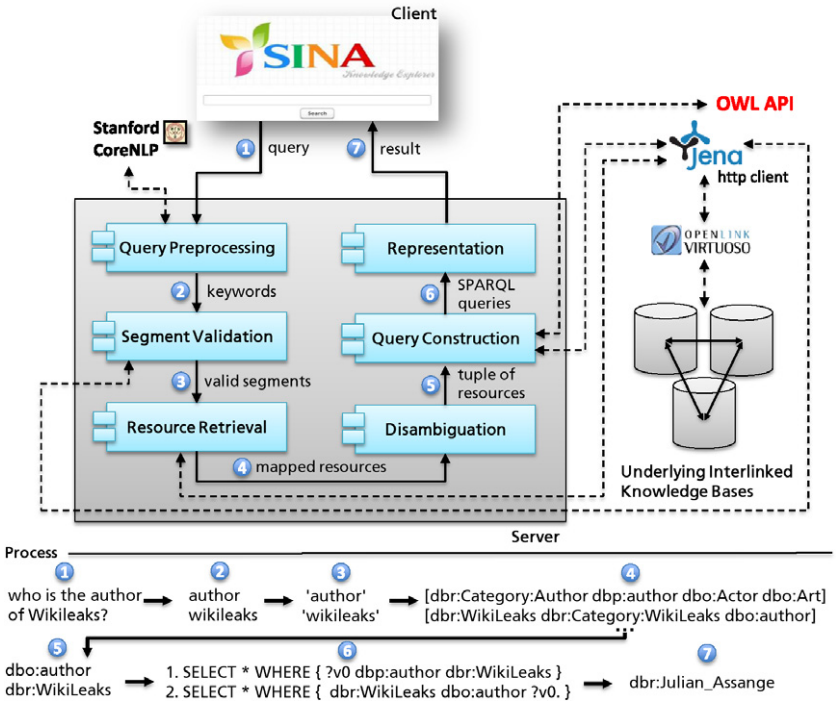

Fig. 1. Architecture of SINA search engine.

the spectrum. Our approach aims to achieve maximal flexibility by being able to generate SPARQL queries from both natural-language queries and keyword queries. This goal is achieved by limiting the type of formal queries (i.e. SPARQL queries) that our approach can generate to conjunctive SPARQL queries. Several challenges need to be addressed to devise such an approach: First, a query segmentation and disambiguation approach for mapping input query to resources has to be devised. To do so, statistical information of the retrieved resources has to be retrieved. Then, a method for generating conjunctive federated SPARQL queries, which can be sent to SPARQL endpoint to retrieve relevant data must be developed.

In this paper, we show how our framework, SINA, implements these different steps. In contrast to previous approaches, SINA can make use of the topology of Linked Data by exploiting links between resources to devise federated SPARQL queries. Thus, it can deal with both retrieving data from either a single dataset or several interlinked datasets. Consequently, it can be used over the whole of the Linked Open Data Cloud. We present a thorough evaluation of our approach on three different datasets: we use the QALD-1 to detect optimal parameters for SINA and present a first evaluation of the approach on this benchmark dataset. We then run SINA on the QALD-3 benchmark and show that we can generate the correct SPARQL queries for 32 of these queries, thus achieving the same results as the best system tested in the benchmark. Finally, we evaluate SINA in a federated scenario against the queries from the life science domain used in [1]. In an effort to make SINA easily portable, we refrained from using dataset-specific indexes and rely fully on the SPARQL endpoint when constructing SPARQL queries. To ensure that our approach still achieves acceptable runtimes, we implemented both a parallel and a sequential version of SINA. In the second part of the evaluation, we thus present a study of SINA's runtime on the QALD-3 benchmark.

This paper is organized as follows: In the subsequent section, we introduce the architecture of the proposed search engine. In Section 3, we present the problem at hand in more detail and some of the notations and concepts used in this work. Section 4 presents the proposed disambiguation method in detail along with the evaluation of the bootstrapping. In Section 5, we then present the key steps of our algorithm for constructing a conjunctive query. Our evaluation results are presented in Section 6 while related work is reviewed in Section 7. We close with a discussion and future work.

2. Overview

In this section, we describe the high-level architecture and implementation of the SINA search engine. Fig. 1 illustrates the architecture of SINA, which comprises six main components. Each component consumes the output of the previous component:

1. The **query preprocessing** component receives the textual input query and applies three functions: (1) Tokenization: extraction of individual keywords, removing punctuation and capitalization. (2) Stop word removal: removal of common words such as articles and prepositions. Since in this version, we do not recognize type of answers, thus wh-questions are removed as stop words. (3) Word lemmatization: determining the lemma of the remaining keywords.
2. The **segment validation** component groups keywords to form segments. This component validates the grouped segments with respect to the available resources in the underlying knowledge base(s).
3. The **resource retrieval** component obtains relevant resources from the underlying knowledge bases. The retrieval is based on the string matching between valid segments and the `rdfs:label` of resources. Furthermore, more resources are inferred from lightweight `owl:sameAs` reasoning.
4. The **disambiguation** component determines the best subset of resources for the given input query.
5. The **query construction** component results in formal queries (i.e. SPARQL query) using the graph-structure of data.
6. The **representation** component shows the retrieved results after evaluating the generated SPARQL queries.

We implemented SINA as a Java web application which is available online. We deployed two demo instances, one employing DBpedia as background knowledge[2] and a second one operating on several interlinked life-science datasets.[3] SINA has a simple interface similar to common search engines. All steps of SINA are carried out automatically and the user does not have to interact with the system during the search process. Moreover, SINA is accessible via api.

3. Problem and preliminaries

In this section, we introduce some crucial notions employed throughout the paper and describe the main challenges that 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 set of keywords by tokenizing, stop-word removal and lemmatization. Our input query thus is an n-tuple of keywords, i.e. $Q = (k_1, k_2, \ldots, k_n)$.

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 $R = \{r_1, r_2, \ldots, r_m\}$. 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 into segments. For each segment, a suitable resource is then to be determined. The challenge here is to determine the right segment granularity, 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 into 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 '*Tuberculosis*' 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 $R = \{r_1, \ldots, r_m\}$ 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.

3.1. Resource disambiguation

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

Definition 1 (*Segment and Segmentation*)**.** For a given query $Q = (k_1, k_2, \ldots, k_n)$, the segment $S_{(i,j)}$ is the sequence of keywords from start position i to end position j, i.e., $S_{(i,j)} = (k_i, k_{i+1}, \ldots, k_j)$. A query segmentation is an m-tuple of segments $SG(Q) = (S_{(0,i)}, S_{(i+1,j)}, \ldots, S_{(l,n)})$ 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.

Definition 2 (*Resource Disambiguation*)**.** Let the segmentation $SG' = (S^1_{(0,i)}, S^2_{(i+1,j)}, \ldots, S^m_{(l,n)})$ be the suitable segmentation for the given query Q. Each segment S^i of SG' is first mapped to a set of candidate resources $R_i = \{r_1, r_2, \ldots, r_h\}$ from the underlying knowledge base. The aim of the disambiguation step is to detect an m-tuple of resources $(r_1, r_2, \ldots, r_m) \in R_1 \times R_2 \times \cdots \times R_m$ 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

[2] http://sina.aksw.org/.

[3] http://sina-linkeddata.aksw.org/.

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

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

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.

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 valid segments.

3.2. 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. $R = (r_1, \ldots, r_m)$. Herein, we consider conjunctive queries being conjunctions 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 a quantifier, thus conjunctive queries to a large extent capture the user information need. A conjunctive query is called query graph and formally defined as follows.

Definition 3 (*Query Graph*)**.** Let a set $R = \{r_1, \ldots, r_n\}$ 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. $QG \equiv \{(n, e, n') | (n, n') \in V^2 \wedge e \in E\}$.

4. 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 high mean reciprocal rank (MRR) above 84% (discussed in Section 4.1).

Constructing the state space: A priori, the state space should be populated with as many states as the total number of 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 3.1). (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 4.1. 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.

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

[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.

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` relation. 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`.

4.1. 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 be 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 [8]. It assigns a hub and an 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 space state as well as the weight of the edge between the two states, into account. The transition probability value decreases while 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 $\sum_{S_i} \Pr(S_j|S_i) = 1$ 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:

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

Here, the probabilities 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 at 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:

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

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 $h = (k_i, k_{i+1}, \ldots, k_j)$ 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:

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

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* [9] 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

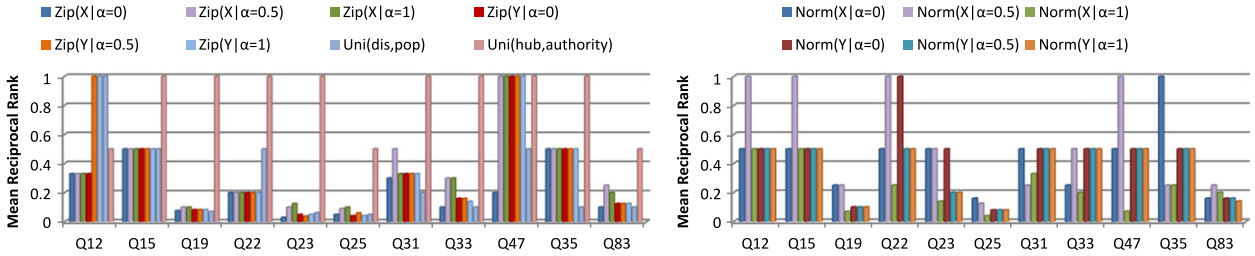

Fig. 2. MRR of different distributions per query for bootstrapping the transition probability.

Table 2
The resources contained in the state space for a given query.

Segment	Resources	Label	Type
Side effect	1. `Sider:sideEffect`	Side effect	Property
	2. `Sider:side_effects`	Side effect	Class
Drug	1. `Drugbank:drugs`	Drug	Class
	2. `Drug bank:offer`	Drug	Class
	3. `Sider:drugs`	Drug	Class
	4. `Diseasome:possibleDrug`	Possible drug	Property
Tuberculosis	1. `Diseasome:1154`	Tuberculosis	Instance
	2. `Sider:C0041296`	Tuberculosis	Instance

have the maximum probability for the respective subsequence 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.

Example 1. Let us consider the query: `What are the side effects of drugs used for Tuberculosis?`
The validated segments are: '`side effect`', '`drug`' and '`Tuberculosis`'. After the retrieval and pruning process, the state space contains the resources listed in Table 2:

By running the *Viterbi algorithm* with the associated probabilities, we have a ranked list of the chosen resources that the sequence {side effect, drug, Tuberculosis} is observable through them. In the following, we show the top-4 most likely paths along with their associated probability.

```
1. 0.0033:    Sider:sideEffect, Diseasome:possibleDrug, Diseasome:1154.
2. 0.0017:    Sider:sideEffect, Diseasome:possibleDrug, Sider:C0041296.
3. 6.0257E-4:   Sider:sideEffect, Sider:drugs,       Diseasome 1154.
4. 4.0805E-4:   Sider:sideEffect, Drugbank:Offer,      Diseasome:1154.
```

4.2. 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

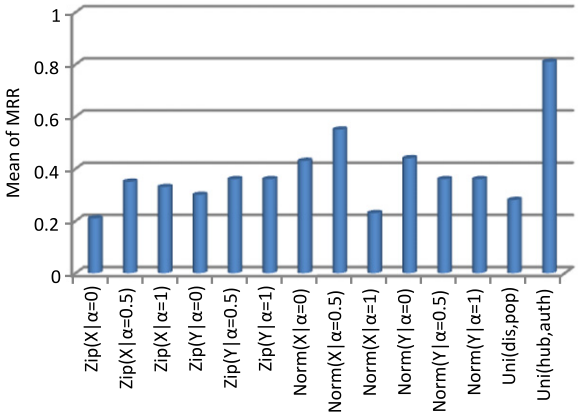

Fig. 3. Comparison of different functions and settings for bootstrapping the transition probability. Uni stands for the uniform distribution, while Zip stands for the Zipfian and Norm for the normal distribution.

authority, we also measured a similar uniform function with the input parameters *distance* and *connectivity degree* defined as:

$$a_{ij} = \frac{distance(S_i - S_j)}{\sum_{\forall S_k > 0} distance(S_i - S_k)} \times connectivitydegree(S_i).$$

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) [10] 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 $\text{MRR}(\mathcal{A}) = \frac{1}{|Q|} \sum_{q_i} \frac{1}{r_i}$. 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] The criterion of choosing the bootstrapped queries was the number of the keywords as well as the associated resources. Fig. 2 shows the MRR achieved by bootstrapping the transition probability of this model with 3 different distribution functions per query

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

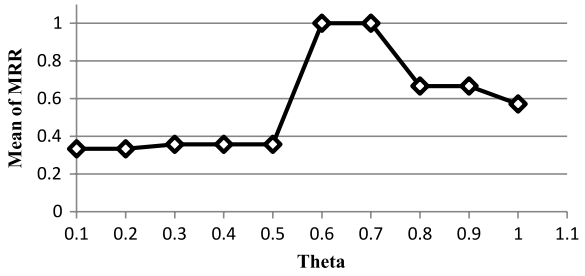

Fig. 4. Mean MRR for different values of θ.

in 14 different settings. Fig. 3 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 trivial in Zipfian and Normal settings, 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.

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. Fig. 4 shows the mean of MRR for different values of the threshold θ employed for pruning the state space. A high value of θ prevents inclusion of some relevant resources 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 0.6 in the rest of our experiments.

5. 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., $R = \{r_1, r_2, \ldots, r_m\}$. 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.

5.1. 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 the product of the relevance probabilities of the n containing triple patterns. We denote the triple patterns with $(s_i, p_i, o_i)_{i=1\ldots n}$ 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 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 find 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.

5.2. 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 node (except nodes connected by `owl:sameAs` links), it eliminates both the need of keeping an

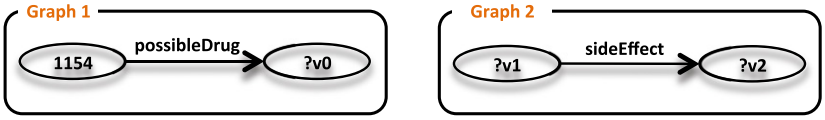

Fig. 5. *IQG* for the Example 2.

index over the neighborhood of nodes, and of using exploration for finding paths between nodes.

Generation of IQGs. After identifying a corresponding set of resources $R = \{r_1, r_2, \ldots, r_m\}$ 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 2. We look at the query: `What are 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 vertex 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`, and (3) an edge labeled `sideEffect` from $?v1$ to $?v2$. Fig. 5 shows the constructed *IQG*, which contains two disjoint graphs.

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 [11], 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 5.1) 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 3. To connect two disjoint graphs i.e. Graph 1 and Graph 2 of the *IQG* shown in Example 2, 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 Fig. 6.

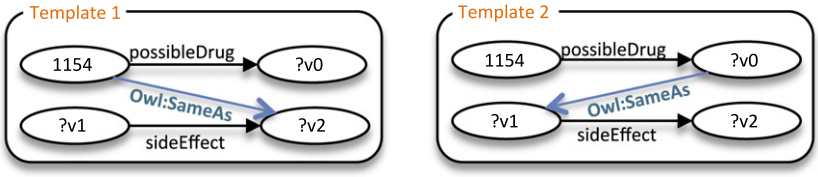

Fig. 6. Generated query graph templates.

6. Evaluation

Experimental setup. The goal of our evaluation was to determine effectiveness and efficiency of (1) the resource disambiguation and (2) the query construction with respect to accuracy and runtime. We employed four different knowledge bases: *DBpedia* as one individual knowledge base and the three interlinked knowledge bases *Drugbank*, *Sider* and *Diseasome*. We measured the effectiveness of our resource disambiguation approach using the *Mean Reciprocal Rank* (MRR). For each query $q_i \in Q$ in the benchmark, we compare the rank r_i assigned by our disambiguation method to the correct m-tuple of resources: $\text{MRR}(\mathcal{A}) = \frac{1}{|Q|}\sum_{q_i}\frac{1}{r_i}$. Moreover, we measured the accuracy of the query construction in terms of precision and recall being defined as follows:

$$Recall = \frac{|correct\ resources\ returned\ by\ generated\ query|}{|resources\ in\ gold\ standard\ answer|}$$

$$Precision = \frac{|correct\ resources\ returned\ by\ generated\ query|}{|all\ resources\ returned\ by\ generated\ query|}.$$

The query construction is initiated with the top-2 tuples returned by the disambiguation. For testing the statistical significance of our results, we used a Wilcoxon signed ranked test with a significance level of 95%.

6.1. Accuracy benchmark over interlinked knowledge bases

To the best of our knowledge, no benchmark for federated queries over Linked Data has been created so far. Thus, we created a benchmark consisting of 25 queries on the three 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. The detailed results of our evaluation are shown in Fig. A.7. We ran our SINA with and without OWL inferencing during the state space construction.

When ran without inferencing, our approach was able to correctly disambiguate 23 out of 25 (i.e. 92%) of the resources contained in the queries. 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). 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.

6.2. Accuracy benchmark over DBpedia

DBpedia [12] the large knowledge base extracted from Wikipedia is an ideal test case with respect to size. There is no standard evaluation benchmark for keyword search over RDF data yet. However, there are the *QALD-1*, *QALD-2* and *QALD-3* benchmarks[10] tailored towards comparing question answering systems for natural language queries[11] [13]. We employed the *QALD-3* test dataset (in order to compare with the recent systems participated in the campaign) and the *QALD-1* dataset (in order to test Sina more) for evaluation (note that *QALD-2* dataset was employed for bootstrapping). Basically, training datasets were used for tuning and debugging Sina. The QALD-3 test dataset (and the QALD-1 benchmark) consist of 100 (respectively 50) questions in natural language that were also formulated as SPARQL queries. The questions are of different levels of complexity. Generally, the reasons for failures are as follows:

1. *Complex questions.* Questions containing quantifiers, comparatives and superlatives.
2. *Coverage*. Questions requiring information from beyond DBpedia ontology, i.e., from `YAGO` or `FOAF`.
3. *Query expansion*. Examples are the keywords "*wife*", which should be matched to "*spouse*" and "*daughter*" to "*child*".
4. *Query cleaning*. An example is the question "*Through which countries does the Yenisei river flow?*" The keyword *flow* is not a stop word but does not have any matched resource in the corresponding SPARQL query of the benchmark; and should therefore be ignored.

Since query expansion and cleaning might result in more noisy input for the model, we did not address these in this work.[12] With respect to the *QALD-3* test dataset, we take all 100 questions in

[8] The benchmark queries are available at http://wiki.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.

[10] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/.

[11] Sina did not participate in the running campaign due to the late preparation, but we used these benchmarks for own evaluation.

[12] A careful extension of our approach and analysis of the results will be required.

English natural language without changes in the original question into account. Accordingly, SINA can correctly answer 32 questions with precision 0.32, recall 0.32, F-measure 0.32 and of these 32 questions, average MRR is 0.87. Thus, it outperforms the most state-of-the-art question answering systems.[13] Note that our approach is limited to conjunctive queries. Regarding the failures, 14 questions were beyond DBpedia coverage; 43 queries had query expansion or query cleaning issues; 7 questions were complex questions. From the remaining questions, in two cases matching to the appropriate resource failed, e.g. due to using abbreviation. In the other two cases, the constructed query does not fulfill the information need of the question. The detailed results of this evaluation are shown in Fig. B.8.

In the sequel, we elaborate on the result of the evaluation using *QALD-1* benchmark. We excluded 7 complex questions and 13 questions requiring information beyond DBpedia, i.e., from `YAGO` and `FOAF`. From the remaining questions 14 require query expansion or query cleaning to map keywords to resources correctly. In order to keep these 14 questions while reducing the effect of external parameters (i.e. expansion and cleaning) in our evaluation analysis, we slightly modified the expression of these 14 questions in such a way that expansion and cleaning is not required anymore. For instance, the query "*Through which countries does the Yenisei river flow?*" was slightly rephrased as `countries of the Yenisei river`.

Fig. C.9 shows the results of the accuracy study for each of the 30 questions used in our evaluation. Only the expression of questions printed with pink background has been modified compared to QALD-1. Out of the 30 questions, our disambiguation method precisely identifies resources for 27 questions with an MRR higher than 96%. Also, the query graph construction method correctly constructs the query graphs retrieving answers for 25 questions exactly as required by the benchmark with precision 0.5, recall 0.5 and F-measure 0.5.

In the following, we discuss the reasons for failures. In the case of Q30, the failure is caused by data quality (i.e., inconsistent domain and range information in DBpedia). Although suitable resources have been identified i.e. `dbp:creator` and `dbr:Goofy`, generating a suitable SPARQL query failed due to a conflict between the domain of the property `dbp:creator` and the type of the entity `dbr:Goofy`. According to the DBpedia schema the domain of the property `dbp:creator` is `dbo:Work` but the type of the entity `dbr:Goofy` has been declared as `dbo:Person`. In other words, there is no intersection between *CT* of `dbr:Goofy` and *OP* of `dbp:creator`. In DBpedia, `dbr:Goofy` is with the type `dbo:Person` wrongly placed as the subject of a triple with property `dbp:creator`. In the case of Q37, the constructed query is not matched to the information need of the question. The constructed query contains the following triples: `1. dbr:Bill-Clinton dbp:child ?c`.

`2. dbr:Bill-Clinton dbp:spouse ?s`. Our approach fails to identify appropriate resources for Q1, Q2 and Q48. This failure is mainly due to the difficulty of recognizing the correct data granularity and consequently identifying the right resources. The question '`When did Germany join the EU?`' needs query expansion. In fact, the segment '`join the EU`' should be mapped to the property `dbp:accessioneudate`. It is very unlikely that a user expresses a query close to the label of that property. In the case of Q1, the segment '`computer software`' should be mapped to a literal, whereas currently the only kind of literal included for looking up is `rdfs:label`. Improving this is part of our extension agenda. Q2 is very complex. For the given segment '`located`' the corresponding SPARQL query contains three triple patterns with the properties `dbo:location`, `dbp:location` and `dbp:locationCountry`. Since we assume that each of the known segments in the benchmark could be mapped to exactly one resource in the SPARQL query, our approach is not yet able to handle such questions. Q30 does not aim at retrieving any resources (it uses `ask` instead of `select`). Although our method successfully identifies correct resources and triple patterns, this question was not considered a correct query.

6.3. Runtime benchmark

Although performance was not (yet) the primary focus of our work, we also wanted to provide evidence that our approach can be used for real-time querying. In the sequential version of SINA, all the requests to the knowledge bases are performed sequentially. Our intuition was that the runtime can be optimized by parallelizing those requests. In other to speedup runtime, we thus implemented parallelization over three components, i.e., *segment validation, resource retrieval* and *query construction*. We evaluated the runtime of our approach on the life-science as well as on QALD-1 benchmark queries. All experiments were carried out on a Windows 7 machine with an Intel Core M 620 processor, 6 GB of RAM and a 230 GB SSD. Table 3 shows the average runtime of SINA over DBpedia and the life science datasets (with and without inferencing during state space construction) for three runs. Through employing parallelization, we achieved a total performance gain of 60% for DBpedia, 16.8% for life science without inferencing and 23.3% for life science with inferencing.

Although most life science queries without inferencing take less time than with inferencing their performance gain is larger. This is due to the fact that the time saved in both versions is almost the same. For instance, the average performance gain for six keyword queries in the life science scenario with and without inferencing is approximately **40** and **33.3** s.

The observed results of runtime show that the number of input keywords is not correlated to the achieved gain or query execution time. There are several explanations: (1) The number of keywords does not directly affect the number of requests sent to the knowledge base. In fact, the number of initial requests sent to the knowledge base for resource retrieval is correlated to the number of valid segments (not input keywords) while the number of valid segments is not depended on the number of input keywords. For instance, for two keywords *A* and *B*, we may have three valid segments as (*A*, *B*, *AB*) while for three keywords *A*, *B* and *C* we may have only two valid segments as (*A*, *BC*). (2) The size of the state space is not related to the size of input keywords. For instance, a keyword *A* can retrieve more resources than a keyword *B*, which means that a query containing keyword *A* will demand more time than a query containing keyword *B*. (3) There are also external factors that affect the query execution runtime such as the knowledge base indexing strategy.

7. Related work

We analyze semantic search approaches in five dimensions, i.e., input query format, disambiguation, expansion, data distribution and query transformation. With respect to the first dimension, there are two common types of input query, i.e., natural language query and keyword query. There is a contradiction in usability studies of these two types of input queries. While [14] shows that users prefer using natural language queries to keywords, [15] presents that students prefer keyword query. Second dimension is using a disambiguation approach which selects the

[13] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/3/qald3_results.pdf.

Table 3
SINA performance in seconds in sequential mode (*Seq*) and with parallelization (*Par*) for different number of keywords (K) and datasets.

#K	DBpedia			Life science			Life science with inf.		
	Seq	*Par*	*%Gain*	*Seq*	*Par*	*%Gain*	*Seq*	*Par*	*%Gain*
2	57.3	17.5	69.3	3.7	3.4	3.4	4.4	3.8	12.0
3	38.3	9.3	75.4	13.4	8.5	36.0	13.5	8.3	38.8
4	30.3	12.5	58.6	68.9	54.9	21.3	104.5	86.4	17.3
5	38.9	18.5	52.3	66.3	59.0	10.9	128.8	112.0	13.0
6	–	–	–	493.1	452.8	8.1	94.7	61.4	35.1
7	39.6	20.5	48.0	–	–	–	–	–	–

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

Fig. A.7. MRR, precision and recall for the life science benchmark.

best interpretation of the input query. Third dimension is query expansion like taking into account synonyms in order to improve retrieval performance. Fourth dimension is related to the number of the underlying knowledge bases, whether the search engine runs on either a single knowledge base or multiple interlinked knowledge bases. The last dimension refers on how to transform the input query to a formal query. Several approaches have been developed for this transformation, i.e., document-centric, entity-centric and question-answering approaches.

Most of these approaches are adaptations of document retrieval approaches. For instance, *Swoogle* [2], *Watson* [3], *Sindice* [4] stick to the document-centric paradigm. Entity-centric approaches (e.g. *Sig.Ma* [16], *Falcons* [17], *SWSE* [18]) 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* [5] and *OntoNL* [6]. *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 cannot handle complex queries. *Pythia* [19] 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* [7], a more flexible question-answering system that combines Pythia's linguistic analysis and the *BOA framework* [20] for detecting properties to natural language patterns. Exploring schema from anchor points bound to input keywords is another approach discussed in [21]. Querying Linked datasets is addressed with the work that mainly treats both the data and queries as bags of words [17,22]. [23] presents a hybrid solution for querying linked datasets. It runs 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. Table 4 compares six Semantic Web search engines in five dimensions. 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 [24]. Segmentation has been studied extensively in the natural language processing (NLP) literature (e.g., [25]). NLP techniques for chunking such as part-of-speech tagging or name entity recognition cannot achieve high performance when applied to query segmentation. [26] 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 [27]. [28] is

Table 4
Comparison of Semantic Web search engines in five dimensions.

Systems	Input query	Disam.	Expan.	Data dis.	Transformation
SINA	NL & Keyword	✓	–	Multiple	Question answering
Sindice	Keyword	–	–	Multiple	Document retrieval
Sigma	Keyword	–	–	Multiple	Entity retrieval
Swoogle	Keyword	–	–	Multiple	Document retrieval
PowerAqua	NL	✓	✓	Multiple	Question answering
TBSL	NL	✓	✓	Single	Question answering

ID	Query	MRR	Pr	Re
2	Who was the successor of John F. Kennedy?	0.5	1	1
15	What is the longest river?	1	1	1
21	What is the capital of Canada?	0.5	1	1
22	Who is the governor of Wyoming?	1	1	1
24	Who was the father of Queen Elizabeth II?	1	1	1
26	How many official languages are spoken on the Seychelles?	1	1	1
30	What is the birth name of Angela Merkel?	1	1	1
33	Give me all Australian nonprofit organizations.	1	1	1
35	Who developed Minecraft?	1	1	1
39	Give me all companies in Munich.	1	1	1
40	List all games by GMT.	1	1	1
41	Who founded Intel?	0.33	1	1
43	Give me all breeds of the German Shepherd dog.	1	1	1
54	What are the nicknames of San Francisco?	1	1	1
56	When were the Hells Angels founded?	1	1	1
57	Give me the Apollo 14 astronauts.	1	1	1
58	What is the time zone of Salt Lake City?	1	1	1
60	Give me a list of all lakes in Denmark.	1	1	1
61	How many space missions have there been?	0.2	1	1
62	Did Socrates influence Aristotle?	1	1	1
63	Give me all Argentine films.	1	1	1
64	Give me all launch pads operated by NASA.	1	1	1
65	Which instruments did John Lennon play?	1	1	1
73	How many children did Benjamin Franklin have?	1	1	1
76	List the children of Margaret Thatcher.	1	1	1
78	Was Margaret Thatcher a chemist?	1	1	1
81	Which books by Kerouac were published by Viking Press?	0.2	1	1
86	What is the largest city in Australia?	1	1	1
87	Who composed the music for Harold and Maude?	1	1	1
88	Which films starring Clint Eastwood did he direct himself?	0.25	1	1
90	Where is the residence of the prime minister of Spain?	1	1	1
100	Who produces Orangina?	1	1	1

Fig. B.8. MRR, precision and recall for QALD-3 questions on DBpedia.

ID	Query	MRR	Pr	Re
29	In which films directed by Garry Marshall was Julia Roberts starring?	1	1	1
46	What is the highest place of Karakoram?	1	1	1
30	Is proinsulin a protein?	1	1	1
33	Who created Goofy?	1	0	0
27	What is the revenue of IBM?	1	1	1
13	In which country is the Limerick Lake?	1	1	1
32	Which television shows were created by Walt Disney?	1	1	1
40	Who is the author of WikiLeaks?	1	1	1
19	What is the currency of the Czech Republic?	1	1	1
11	What is the area code of Berlin?	1	1	1
16	Who is the owner of Universal Studios?	1	1	1
26	Give me all soccer clubs in Spain.	1	1	1
43	Which river does the Brooklyn Bridge cross?	1	1	1
5	What are the official languages of the Philippines?	1	1	1
47	Death cause of Bruce Carver.	1	1	1
7	Death place of Abraham Lincoln.	1	1	1
49	Height of Claudia Schiffer.	1	1	1
8	Date of Battle of Gettysburg.	0.5	1	1
34	Countries of the Yenisei river.	1	1	1
15	Occupation of Frank Herbert.	1	1	1
12	Classis of the Millepede.	1	1	1
31	Museum of The Scream.	1	1	1
50	Source country of Nile.	1	1	1
10	Spouse of Barak Obama.	1	1	1
2	Which telecommunications organizations are located in Belgium?	0	0	0
6	Name of leader of New York City.	0.5	1	1
41	Designer of Brooklyn Bridge?	1	1	1
1	Which companies are in the computer software industry?	0	0	0
37	Spouse of child of Bill Clinton.	1	0	0
48	When did Germany join the EU?	0	0	0

Fig. C.9. MRR, precision and recall for QALD-1 questions on DBpedia.

a supervised method based on Conditional Random Fields (CRF) whose parameters are learned from query logs. For detecting named entities, [29] 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 [30–33]. Still, the most common approach is using the context for disambiguation [34–36]. 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.

8. Discussion and conclusion

The result of evaluation shows the effectiveness as well as scalability of this approach. In the current implementation forward chaining is carried out on the fly. Consequently, the runtime can be further significantly increased by pre-processing the knowledge base, adding all statements that can be generated via forward chaining and constructing an index for the label information. After implementing further performance optimizations (e.g. caching computed values such as resource distances), we expect our implementation to require less than 10 s for up to 5 keywords. Note that 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).

Appendix A

See Fig. A.7.

Appendix B

See Fig. B.8.

Appendix C

See Fig. C.9.

References

[1] S. Shekarpour, A.-C.N. Ngomo, S. Auer, Question answering on interlinked data, in: D. Schwabe, V.A.F. Almeida, H. Glaser, R.A. Baeza-Yates, S.B. Moon (Eds.), WWW, International World Wide Web Conferences Steering Committee, ACM, 2013, pp. 1145–1156.

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

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

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

[5] V. Lopez, M. Fernández, E. Motta, N. Stieler, PowerAqua: Supporting users in querying and exploring the Semantic Web, J. Semant. Web 3 (3) (2012) 249–265.

[6] A. Karanastasi, A. Zotos, S. Christodoulakis, The OntoNL Framework for Natural Language Interface Generation and a Domain-Specific Application, in: DELOS Conference, 2007, pp. 228–237.

[7] C. Unger, L. Bühmann, J. Lehmann, A.-C.N. Ngomo, D. Gerber, P. Cimiano, Template-Based Question Answering Over RDF Data, ACM, 2012.

[8] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632.

[9] A.J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans. Inform. Theory (1967).

[10] E. Vorhees, The TREC-8 question answering track report, in: Proceedings of TREC-8, 1999.

[11] D.R. Cheriton, R.E. Tarjan, Finding minimum spanning trees, SIAM J. Comput. 5 (4) (1976) 724–742.

[12] J. Lehmann, C. Bizer, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia—a crystallization point for the web of data, J. Web Semant. 7 (3) (2009) 154–165. http://dx.doi.org/10.1016/j.websem.2009.07.002. URL: http://jens-lehmann.org/files/2009/dbpedia_jws.pdf.

[13] V. Lopez, C. Unger, P. Cimiano, E. Motta, Evaluating question answering over linked data, J. Web Semant. 21 (2013) 3–13.

[14] E. Kaufmann, A. Bernstein, How useful are natural language interfaces to the semantic web for casual end-users?, in: The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, 2008, pp. 281–294.

[15] M. Reichert, S. Linckels, C. Meinel, T. Engel, Student's perception of a semantic search engine, in: CELDA, IADIS, 2005, pp. 139–147.

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

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

[18] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, S. Decker, Searching and browsing linked data with SWSE: the semantic web search engine, J. Web Semant. 9 (4) (2011) 365–401.

[19] C. Unger, 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, 2011, pp. 153–160.

[20] D. Gerber, A.-C. Ngonga Ngomo, Extracting multilingual natural-language patterns for RDF predicates, in: Proceedings of EKAW, 2012.

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

[22] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, Y. Pan, Semplore: A scalable IR approach to search the Web of Data, J. Web Semant. 7 (3) (2009) 177–188.

[23] D.M. Herzig, T. Tran, Heterogeneous Web Data Search Using Relevance-Based on the Fly Data Integration, ACM, 2012, pp. 141–150.

[24] A. Uzuner, B. Katz, D. Yuret, Word Sense Disambiguation for Information Retrieval, AAAI Press, 1999.

[25] L.A. Ramshaw, M.P. Marcus, Text chunking using transformation-based learning, CoRR.

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

[27] B. Tan, F. Peng, Unsupervised query segmentation using generative language models and wikipedia, in: WWW, ACM, 2008.

[28] X. Yu, H. Shi, Query Segmentation Using Conditional Random Fields, ACM, 2009.

[29] J. Guo, G. Xu, X. Cheng, H. Li, Named Entity Recognition in Query, ACM, 2009.

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

[31] B. Tan, F. Peng, Unsupervised Query Segmentation Using Generative Language Models and Wikipedia, ACM, 2008.

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

[33] A. Shepitsen, J. Gemmell, B. Mobasher, R. Burke, Personalized Recommendation in Social Tagging Systems Using Hierarchical Clustering, ACM, 2008.

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

[35] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, E. Ruppin, Placing search in context: the concept revisited, in: WWW, 2001.

[36] R. Kraft, C.C. Chang, F. Maghoul, R. Kumar, Searching with context, in: WWW'06, ACM, 2006.