Query Segmentation and Resource Disambiguation Leveraging Background Knowledge

Saeedeh Shekarpour[1], Axel-Cyrille Ngonga Ngomo[1], and Sören Auer[1]

Department of Computer Science, University of Leipzig
Johannisgasse 26, 04103 Leipzig
{lastname}@informatik.uni-leipzig.de

Abstract. Accessing the wealth of structured data available on the Data Web is still a key challenge for lay users. Keyword search is the most convenient way for users to access information (e.g., from data repositories). In this paper we introduce a novel approach for determining the correct resources for user-supplied keyword queries based on a hidden Markov model. In our approach the user-supplied query is modeled as the observed data and the background knowledge is used for parameter estimation. Instead of learning parameter estimation from training data, we leverage the semantic relationships between data items for computing the parameter estimations. In order to maximize accuracy and usability, query segmentation and resource disambiguation are mutually tightly interwoven. First, an initial set of potential segmentations is obtained leveraging the underlying knowledge base; then the final correct set of segments is determined after the most likely resource mapping was computed using a scoring function. While linguistic methods like named entity, multi-word unit recognition and POS-tagging fail in the case of an incomplete sentences (e.g. for keyword-based queries), we will show that our statistical approach is robust with regard to query expression variance. Our experimental results when employing the hidden Markov model for resource identification in keyword queries reveal very promising results.

1 Introduction

The Data Web currently amounts to more than 31 billion triples[1] and contains a wealth of information on a large number of different domains. Yet, accessing this wealth of structured data *remains* a key challenge for lay users. The same problem emerged in the last decade when users faced the huge amount of information available of the Web. Keyword search has been employed by popular Web search engines to provide access to this information in a user-friendly, low-barrier manner. Meanwhile, keyword search has become one of the most convenient way for users to access information on the Web [23]. The large amount of research on

[1] See http://www4.wiwiss.fu-berlin.de/lodcloud/state/ (May 23th, 2012)

the successful application of keyword-based search in document retrieval and the acknowledged usability of this paradigm are convincing reasons for employing the keyword search paradigm also on the Data Web. However, keyword search in structured data raises three main difficulties:

- First, the *right segments of data items* that occur in the keyword queries have to be identified. For example, the query *'Who produced films starring Natalie Portman'* can be segmented to (*'produce'*, *'film'*, *'star'*, *'Natalie Portman'*) or (*'produce'*, *'film star'*, *'Natalie'*, *'Portman'*). Note that the first segmentation is more likely to lead to a query that contain the results intended for by the user.
- Second, these segments have to be disambiguated and mapped to the right resources. Note that the resource ambiguity problem is of increasing importance as the size of knowledge bases on the Linked Data Web grows steadily. Considering the previous example[2], the segment *'film'* is ambiguous because it may refer to the class `dbo:Film` (the class of all movies in DBpedia) or to the properties `dbo:film` or `dbp:film` (which relates festivals and the films shown during these festivals).
- Finally, adequate SPARQL queries have to be generated based on these resources.

While the third step has been addressed in previous works [23, 14, 19], only little attention has been paid to the first two steps in the context of keyword search on Linked Data. Yet, these two steps play a crucial role in the querying of structured data as a wrong segmentation mostly leads to a wrong resource identification, which itself induces the computation of erroneous SPARQL queries.

In this paper, we present an automatic query segmentation and resource disambiguation approach leveraging background knowledge. The initial *query segmentation* step of our approach transforms input query words into segments in either a naive or a greedy fashion. These segments are then mapped to candidate sets of resources found in the knowledge base (essentially properties, instances and classes). A single query segment is mapped to multiple resources and identification of the correct resource is ambiguous. We identify the correct resources from the various candidate resources associated with each segment of a given query which will be referred to as *resource disambiguation*. Determining the combination of resources is then carried out by using a hidden Markov model, where the user-supplied query is modeled as the observed data and the background knowledge is used for parameter estimation. In order to maximize accuracy and usability, query segmentation and resource disambiguation are mutually tightly interwoven. Our approach returns a list of segmentations and resource mappings ranked by a scoring function based on the likelihood of the segmentation and disambiguation based on the observed data. Note that we do not rely on training data for the parameter estimation. Instead, we leverage the semantic relationships between data items for this purpose. While linguistic methods like

[2] The underlying knowledge base and schema used throughout the paper for examples and evaluation is DBpedia 3.7 dataset and ontology.

named entity, multi-word unit recognition and POS-tagging fail in the case of an incomplete sentences (e.g. for keyword-based queries), we will show that our statistical approach is robust with regard to query expression variance.

This article is organized as follows: We review related work in Section 2. In Section 3 we present formal definitions laying the foundation for our work. In the section 4 our approach is discussed in detail. For a comparison with natural language processing (NLP) approaches section 5 introduces an NLP approach for segmenting query. Section 6 presents experimental results. In the last section, we close with a conclusion and an outlook on potential future work.

2 Related Work

Our approach is mainly related to two areas of research: text and query segmentation and entity disambiguation. Text segmentation has been studied extensively in the natural language processing (NLP) literature, and includes tasks such as noun phrase chunking where the task is to recognize the chunks that consist of noun phrases (see e.g., [17]). Yet, such approaches cannot be applied to query segmentation since queries are short and composed of keywords. Consequently, NLP techniques for chunking such as part-of-speech tagging [4] or name entity recognition [7, 5] cannot achieve high performance when applied to query segmentation. Segmentation methods for document-based Information Retrieval can be categorized into statistical and non-statistical algorithms. As an example of none statistical methods, [16] addresses the segmentation problem as well as spelling correction. Each keyword in a given query is first expanded to a set of similar tokens in the database. Then, a dynamic programming algorithm is used to search for the segmentation based on a scoring function. The statistical methods fall into two groups, namely supervised and unsupervised methods. For example, the work presented in [21] proposes an unsupervised approach to query segmentation in Web search. Yet this technique can not be easily applied to structured data. Supervised statistical approaches are used more commonly. For example, [28] presents a principled statistical model 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 Webpages, search result snippets and Wikipedia titles have been used [18, 22, 3]. Current segmentation algorithms are not applicable to our segmentation problem for several reasons. First, because they mostly are not intended for search on structured data, it is not guaranteed that the segments they retrieve are actually part of the underlying knowledge base. Another problem with these segmentation algorithms is that they ignore the semantic relationships between segments of a segmentation. Thus, they are likely to return sub-optimal segmentations.

An important challenge in Web search as well as in Linked Data Search is entity disambiguation. Keyword queries are usually short and inherently lead to significant keyword ambiguity as one query word may represent different information needs for different users [24]. There are different ways for tackling this

```
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.

challenge; firstly, query clustering [6, 26, 2] applies unsupervised machine learning to cluster similar queries. The basic insight here is that it has been observed that users with similar information needs click on a similar set of pages, even though the queries they pose may vary. Other approaches apply query disambiguation, which tries to find the most appropriate sense of each keyword. To achieve this goal, one way is involving the user in selecting the correct sense [11, 10, 27]. Another technique for disambiguation is personalized search by using a history of the user activities to tailor the best choice for disambiguation [1, 20, 29]. Still, the most common approach is using context for disambiguation [15, 8, 13]. Albeit context has been defined vaguely (with various definitions), herein we define context as information surrounding the given query which can be employed for augmenting search results. In this work, resource disambiguation is based on a different type of context: we employ the structure of the knowledge at hand as well as semantic relations between the candidate resources mapped to the possible segmentations of the input query.

3 Formal Specification

RDF data is modeled as a directed, labeled graph $G = (V, E)$ where V is a set of nodes i.e. the union of entities and property values, and E is a set of directed edges i.e. the union of object properties and data value properties. The user-supplied query can be either a complete or incomplete sentence. However, after removing the stop words, typically set of keywords remains. The order in which keywords appear in the original query is partially significant. Our approach can map adjacent keywords to a joint resource. 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 reasonable assumption, since users will write strongly related keywords together, while the order of only loosely related keywords or keyword segments may vary. The input query is formally defined as an n-tuple of keyword, i.e. $Q = (k_1, k_2, ..., k_n)$. We aim to transform

the input keywords into a suitable set of entity identifiers, i.e. resources $R = \{r_1, r_2...r_m\}$. In order to accomplish this task the input keywords have to be grouped together as segments and for each segment a suitable resource should be determined.

Definition 1 (Segment and Segmentation). *For a given query Q, a segment $S_{(i,j)}$ is a sequence of keywords from start position i to end position j which is denoted as $S_{(i,j)} = (k_i, k_{i+1}, ..., k_j)$. A query segmentation is an m-tuple of segments $SG_q = (S_{(0,i)}, S_{(i+1,j)}, ..., S_{(m,n)})$ where the segments do not overlap with each other and 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). *Lets the segmentation $SG' = (S^1_{(0,i)}, S^2_{(i+1,j)}, ..., S^x_{(m,n)})$ be a suitable segmentation for the given query Q. Each segment is mapped to multiple candidate resources from the underlying knowledge base, i.e. $S^i \rightarrow R^i = \{r_1, r_2...r_h\}$. The aim of disambiguation is to choose an x-tuple of resources from the Cartesian product of sets of candidate resources $(r_1, r_2, ..., r_x) \in \{R^1 \times R^2 \times ...R^x\}$ 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 x-tuple.*

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 it includes the next keyword into the current segment as a new segment and checks whether adding the new keyword would make the new segment no longer valid. We repeat this process until we reach the end of the query. Algorithm 2 shows a greedy approach for generating segments. It also starts with the first keyword in the given query as the first segment but keeps including the next keywords into the current segment until adding the new keyword would make the current segment no longer valid. Then, another segment starts from the current position. As a running example, lets assume the input query is *'Give me all video games published by Mean Hamster Software'*. Table 1 shows the set of valid segments based on naive algorithm along with some samples of the candidate resources.

Resource Disambiguation using a ranked list of Cartesian product tuples. A naive approach for finding the correct $x - tuple$ of resources is using a ranked list of tuples from the Cartesian product of sets of candidate resources $\{R^1 \times R^2 \times ...R^n\}$. The n-tuples from the Cartesian product are simply sorted based on the aggregated relevance score (e.g. similarity and popularity) of all contained resources.

4 Query Segmentation and Resource Disambiguation using Hidden Markov Models

In this section we describe how hidden Markov models are used for query segmentation and resource disambiguation. First we introduce the concept of hidden Markov models and then we detail how we define the parameters of a hidden Markov model for solving the query segmentation and entity disambiguation problem.

4.1 Hidden Markov Models

The Markov model is a stochastic model containing a set of states. The process of moving from one state to another state generates a sequence of states. The probability of entering each state only depends on the previous state. This memoryless property of the model is called *Markov property*. Many real-world processes can be modeled by Markov models. A hidden Markov model is an extension of the Markov model, which allows the observation symbols to be emitted from each state with a finite probability. The main difference is that by looking at the observation sequence we cannot say exactly what state sequence has produced these observations; thus, the state sequence is *hidden*. However, the probability of producing the sequence by the model can be calculated as well as which state sequence was most likely to have produced the observations.

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

- X is a finite set of states, Y denotes the set of observed symbols;
- $A : X \times X \rightarrow \mathbb{R}$ is the transition matrix that each entry $a_{ij} = Pr(S_j|S_i)$ shows the transition probability from state i to state j;
- $B : X \times Y \rightarrow \mathbb{R}$ represents the emission matrix, in which each entry $b_{ih} = Pr(h|S_i)$ is associated with the probability of emitting the symbol h from state i;
- π denoting the initial probability of states $\pi_i = Pr(S_i)$.

Data: q: n-tuple of keywords, knowledge base
Result: SegmentSet: Set of segments

```
1  SegmentSet=new list of segments;
2  start=1;
3  while start <= n do
4  |  i = start;
5  |  while S_(start,i) is valid do
6  |  |  SegmentSet.add(S_(start,i));
7  |  |  i++;
8  |  end
9  |  start=i;
10 end
```

Algorithm 2: Greedy algorithm for determining valid segments taking the order of keywords into account.

Segments	Samples of Candidate Resources
video	1. dbp:video
video game	1. dbo:VideoGame
game	1. dbo:Game 2. dbo:games 3. dbp:game 4. dbr:Game 5. dbr:Game_On
publish	1. dbo:publisher 2. dbp:publish 3. dbr:Publishing
mean	1. dbo:meaning 2. dbp:meaning 3. dbr:Mean 4. dbo:dean
mean hamster	1. dbr:Mean_Hamster_Software
mean hamster software	1. dbr:Mean_Hamster_Software
hamster	1. dbr:Hamster
software	1. dbo:Software 2. dbp:software

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

4.2 State Space and Observation Space

State Space. A state represents a knowledge base entity. Each entity has an associated *rdfs:label* which we use to label the states. The actual number of states X is potentially high because it contains theoretically all RDF resources, i.e. $X = V \cup E$. However, in practice we limit the state space by excluding irrelevant states. A relevant state is defined as a state for which a valid segment can be observed. In other words, a valid segment is observed in an state if the probability of emitting that segment is higher than a certain threshold θ. The probability of emitting a segment from a state is computed based on a similarity scoring which we describe in the section 4.3. Therefore, the state space of the model is pruned and contains just a subset of resources of the knowledge base, i.e. $X \subset V \cup E$. In addition to these candidate states, we add an **unknown entity state** to the set of states. The *unknown entity* (UE) state comprises all entities, which are not available (anymore) in the pruned state space.

Observation Space. The observation space is the set of all valid segments found in the input user query (using e.g. the Algorithm 1). It is formally is defined as $O = \{o | \text{o is a valid segment}\}$.

4.3 Emission Probability

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

- The *set-similarity level* measures the difference between the label and the segment in terms of the number of words using the the *Jaccard similarity*.
- 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*.

Our similarity scoring method is now 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:

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

This formula is essentially an extension of the *Jaccard similarity coefficient*. The difference is that in the numerator, instead of using the cardinality of intersections the sum of the string-similarity score of the intersections is computed. 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 have been down-weighted by the factor 0.1 to reduce their influence (since they do not convey much meaningful information).

4.4 Hub and Authority of States

Hyperlink-Induced Topic Search (HITS) is a link analysis algorithm for ranking Web pages [12]. It assigns a hub value and an authority value to each Web page. The *hub value* for each page estimates the value of its links to other pages and the *authority value* estimates the value of the content of a page. Authority and hub values are defined in terms of one another 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 is graph-structured data and entities are linked together, we employed a weighted version of the HITS algorithm in order to assign different popularity values to the states in the state space. For each state we assign a hub value and an authority value. A good hub state is one that points to many good authority states and a good authority state is one that is pointed to from many good hub states. Before discussing the HITS computations, we define the edges between the states in the HMM. For each two states i and j in the state space, we add an edge if there is a path in the knowledge base between the two corresponding resources of maximum length k. Note, that we also take property resources into account when computing the path length.The path length between resources in the knowledge base is assigned as weight to the edge between corresponding states. We use a weighted version of the HITS algorithm to take the distance between states into account. The authority of a state is computed as:

$$\text{For all } S_i \in S \text{ which point to } S_j : auth_{S_j} = \sum_{\forall i} w_{i,j} * hub_{S_i}$$

And the hub value of a state is computed as:

$$\text{For all } S_i \in S \text{ which are pointed to by } S_j : hub_{S_j} = \sum_{\forall i} w_{i,j} * auth_{S_i}$$

The weight $w_{i,j}$ is defined as $w_{i,j} = k - pathLength(i,j)$, where $pathLength(i,j)$ is the length of the path between i and j. These definitions of hub and authority for states are the foundation for computing the transition probability in the underlying hidden Markov model.

4.5 Transition Probability

As mentioned in the previous section, each edge between two states shows the shortest path between them with the length less or equal to k-hop. The edges are weighted by the length of the path. Transition probability shows the probability of going from state i to state j. For computing the transition probability, we take into account the connectivity of the whole of space state as well as the weight of the edge between two states. The transition probability values decrease with the distance of the states, e.g. transitions between entities in the same triple have higher probability than transitions between entities in triples connected through extra intermediate entities. In addition to the edges recognized as the shortest path between entities, there is an edge between each state and the *Unknown Entities* state. The transition probability of state j following state i denoted as $a_{ij} = Pr(S_j|S_i)$. For each state i the condition $\sum_{\forall S_j} Pr(S_j|S_i) = 1$ should be held. The transition probability from the state i to *Unknown Entity* (UE) state is defined as:

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

And means a good hub has less probability to go to UE state. Thereafter, the transition probability from the state i to state j is computed as:

$$a_{ij} = Pr(S_j|S_i) = \frac{auth_{S_j}}{\sum_{\forall a_{ik}>0} auth_{S_k}} * hub_{S_i}$$

Here, the edges with the low distance value and higher authority values are more probable to be met.

4.6 Initial Probability

The initial probability π_{S_i} is the probability that the model assigns to the initial state 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 $InitialStates$. If we assume π_{UE} equals β, then the π_{S_i} of the initial states are defined as follows:

$$\pi_{S_i} = \frac{auth_{S_i} + hub_{S_i}}{\sum_{\forall S_j \in InitialStates} (auth_{S_j} + hub_{S_j})} * (1 - \beta)$$

In fact, π_{S_i} of an initial state depends on both hub and authority values. Figure 1 illustrates an instantiated hidden markov model. The set of hidden states are represented by circles. The state UE refers to the absent resources

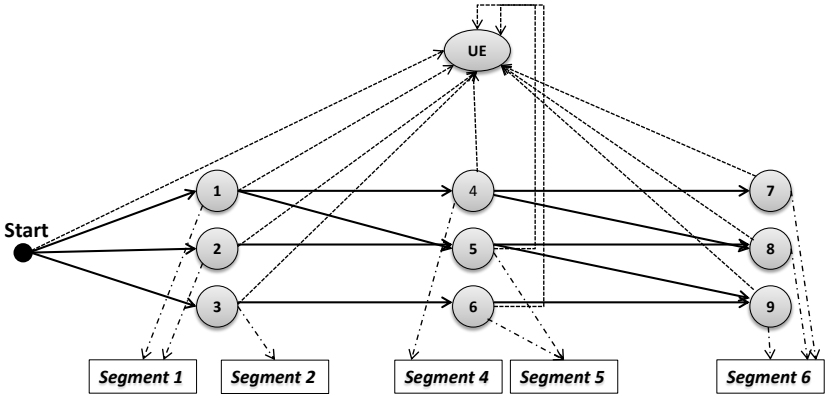


Fig. 1. Trellis representation of Hidden Markov Model.

in the model and other hidden states are relevant resources. Each segment box represents a possible observation. The arrows show a transition from one state to another state and the dashed arrows shows an emitted observation associated with a specific state.

4.7 Viterbi Algorithm for the K-best Set of Hidden States

The optimal path through the markov model for a given sequence (i.e. input query keywords) reveals disambiguated resources forming a correct segmentation. The *Viterbi algorithm* or *Viterbi path* is a dynamic programming approach for finding the optimal path through the markov model for a given 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 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. Therefore, in our implementation we 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: {*dbo:VideoGame, dbo:publisher, dbr:Mean-Hamster-Software*} and consequently the reduced set of valid segments is: {*VideoGam, publisher, Mean-Hamster-Software*} .

5 Query Segmentation using Natural Language Processing

Natural language processing (NLP) techniques are commonly used for text segmentation. Here, we use a combination of named entity and multi-word unit

recognition services as well as POS-tagging for segmenting the input-query. In the following, we discuss this approach in more detail.

Detection of Segments: Formally, the detection of segments aims to transform the set of keywords $K = \{k_1, .., k_n\}$ into a set of segments $\mathcal{T} = \{t_1, ..., t_m\}$ where each k_i is a substring of exactly one $t_j \in \mathcal{T}$. Several approaches have already been developed for this purpose, each with its own drawbacks: Semantic lookup services (e.g., *OpenCalais*[3] and *Yahoo! SeoBook*[4] as used in the current implementation) allow to extract *named entities* (NEs) and *multi-word units* (MWUs) from query strings. While these approaches work well for long queries such as *"Films directed by Garry Marshall starring Julia Roberts"*, they fail to discover noun phrases such as "highest place" in the query *"Highest place of Karakoram"*. We remedy this drawback by combining lookup services and a simple noun phrase detector based on POS tags. This detector first applies a POS tagger to the query. Then, it returns all sequences of keywords whose POS tags abide by the following right-linear grammar:

1. $S \rightarrow adj\ A$ 2. $S \rightarrow nn\ B$ 3. $A \rightarrow B$
4. $B \rightarrow nn$ 5. $B \rightarrow nn\ B$

where S is the start symbol, A and B are non-terminal symbols and nn (noun) as well as adj (adj) are terminal symbols. The compilation of segments is carried as follows: We send the input K to the NE and MWU detection services as well as to the noun phrase detector. Let $\mathcal{N}$ be the set of NEs, $\mathcal{M}$ the set of MWUs and $\mathcal{P}$ the set of noun phrases returned by the system. These three sets are merged to a set of labels $\mathcal{L} = (\mathcal{N} \oplus \mathcal{M}) \oplus \mathcal{P}$, where $\oplus$ is defined as follows:

$$A \oplus B = A \cup B \backslash \{b \in B | \exists a \in A\ overlap(a, b)\}, \quad (1)$$

where $overlap(a, b)$ is true if the strings a and b overlap. The operation $\oplus$ adds the longest elements of B to A that do not overlap with A. Note that this operation is not symmetrical and prefers elements of the set A over those of the set B. For example, "`river which Brooklyn Bridge crosses`" leads to $\mathcal{N}$ = {"`Brooklyn Bridge`"}, M = {"`Brooklyn`" , "`Brooklyn Bridge`"} and $\mathcal{P}$ = {"`Brooklyn Bridge`"}. Thus, $\mathcal{L} = (\mathcal{N} \oplus \mathcal{M}) \oplus \mathcal{P}$ = {"`Brooklyn Bridge`"}. The final set of segments $\mathcal{T}$ is computed by retrieving the set of all single keywords that were not covered by the approaches above and that do not occur in a list of stopwords. Thus, for the query above, $\mathcal{T}$ = {"Brooklyn Bridge", "river", "cross"}.

6 Evaluation

The goal of our experiments was to measure the accuracy of resource disambiguation approaches for generating adequate SPARQL queries. Thus, the main question behind our evaluation was as follows: Given a keyword-based query(KQ)

[3] http://viewer.opencalais.com/
[4] http://tools.seobook.com/yahoo-keywords/

or a natural-language query (NL) and the equivalent SPARQL query, how well do the resources computed by our approaches resemble the gold standard. It is important to point out that a single erroneous segment or resource can lead to the generation of a wrong SPARQL query. Thus, our criterion for measuring the correctness of segmentations and disambiguations was that *all of the recognized segments* as well as *all of the detected resources* had to match the gold standard.

6.1 Experimental Setup

So far, no benchmark for query segmentation and resource disambiguation has been proposed in literature. Thus, we created such a benchmark from the DBpedia fragment of the question answering benchmark *QALD-2*[5]. The QALD-2 benchmark data consists of 100 training and 100 test questions in natural-language that are transformed into SPARQL queries. In addition, it contains a manually created keyword-based representation of each of the natural-language questions. The benchmark assumed the generic query generation steps for question answering: First, the correct segments have to be computed and mapped to the correct resources. Then a correct SPARQL query has to be inferred by joining the different resources with supplementary resources or literals. As we are solely concerned with the first step in this paper, we selected 50 queries from the QALD-2 benchmark (25 from the test and 25 from the training data sets) that were such that each of the known segments in the benchmark could be mapped to exactly one resource in the SPARQL query and vice-versa. Therewith, we could derive the correct segment to resource mapping directly from the benchmark[6]. Queries that we discarded include *"Give me all soccer clubs in Spain"*, which corresponds to a SPARQL query containing the resources {`dbo:ground, dbo:SoccerClub, dbr:Spain` }. The reason for discarding this particular query was that the resource `dbo:ground` did not have any match in the list of keywords. Note that we also discarded queries requiring schema information beyond DBpedia schema. Furthermore, 6 queries out of the 25 queries from the training data set[7] and 10 queries out of 25 queries from the test data set[8] required a query expansion to map the keywords to resources. For instance, the keyword *"wife"* should be matched with *"spouse"* or *"daughter"* to *"child"*.

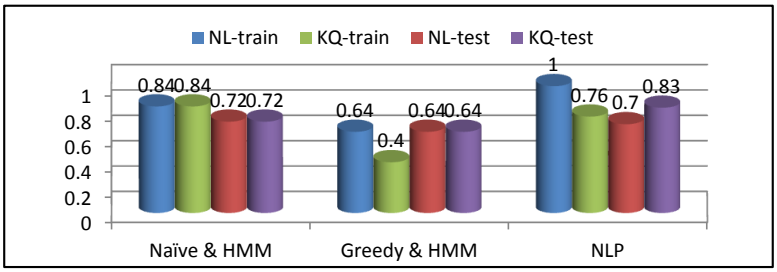
Given that the approaches at hand generate and score several possible segmentations (resp. resource disambiguation), we opted for measuring the *mean reciprocal rank* MRR [25] for both the query segmentation and the resource disambiguation tasks. For each query $q_i \in Q$ in the benchmark, we compare the rank r_i assigned by different algorithms to the correct segmentation and to the resource disambiguation: $MRR(\mathcal{A}) = \frac{1}{|Q|} \sum_{q_i} \frac{1}{r_i}$. Note that if the correct segmentation (resp. resource disambiguation) was not found, the reciprocal rank is

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

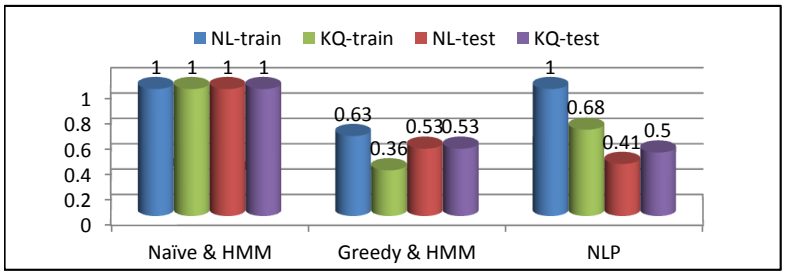
[6] The queries and result of the evaluation and source code is available for download at http://aksw.org/Projects/lodquery

[7] Query IDs: 3, 6, 14, 43, 50, 93.

[8] Query IDs: 3, 20, 28, 32, 38, 42, 46, 53, 57, 67.



(a) Queries that require query expansion are included.



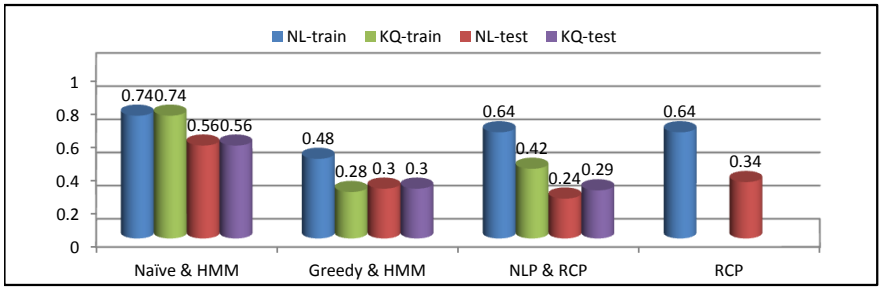
(b) Queries that require query expansion are not included.

Fig. 2. Mean reciprocal rank of query segmentation (first stage).

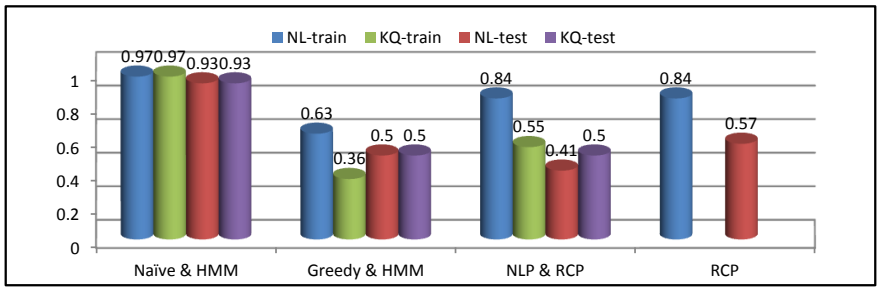
assigned the value 0. At last, we should mention that the employed setting for the parameter k as the number of hops is 6, β as the initial probability to the unknown entity (UE) state is 0 and the threshold θ for punning the state space is equal to 0.7.

6.2 Results

We evaluated our hidden Markov model for resource disambiguation by combining it with the naive (Naive & HMM) and the greedy segmentation (Greedy & HMM) approaches for segmentation. We use the natural language processing (NLP) approach as a baseline in the segmentation stage. For the resource disambiguation stage, we combine ranked Cartesian product (RCP) with the natural language processing (NLP & RCP) and manually injected the correct segmentation (RCP) as the baseline. Note that we refrained from using any query expansion method. The segmentation results are shown in Figure 2. The MRR are computed once with the queries that required expansion and once without. Figure 2(a), including queries requiring expansion, are slightly in favor of NLP, which achieves on overage a 4.25% higher MRR than Naive+HMM and a 24.25% higher MRR than Greedy+HMM. In particular, NLP achieves optimal scores when presented with the natural-language representation of the queries from the "train" data set. Naive+HMM clearly outperforms Greedy+HMM in all settings. The main reason for NLP outperforming Naive+HMM with respect to the seg-



(a) Queries that require query expansion are included.



(b) Queries that require query expansion are not included.

Fig. 3. Mean reciprocal rank of resource disambiguation (second stage).

mentation lies in the fact that Naive+HMM and Greedy+HMM are dependent on matching segments from the query to resources in the knowledge base (i.e. segmentation and resource disambiguation are interwoven). Thus, when no resource is found for a segment (esp. for queries requiring expansion) the HMM prefers an erroneous segmentation, while NLP works independent from the disambiguation phase. However, as it can be observed NLP depends on the query expression. Figure 2(b) more clearly highlights the accuracy of different approaches. Here, the MRR without queries requiring expansion is shown. Naive+HMM perfectly segments both natural language and keyword-based queries. The superiority of intertwining segmentation and disambiguation in Naive+HMM is clearly shown by our disambiguation results in the second stage in Figure 3. In this stage, Naive+HMM outperforms Greedy+HMM, NLP+RCP and RCP in all four experimental settings. Figure 3(a) shows on average 24% higher MRR, although queries requiring expansion are included. In the absence of the queries that required an expansion (Figure 3(b)), Naive+HMM on average by 38% superior to all other approaches and 25% superior to RCP. Note that RCP relies on correct segmentation which in reality is not always a valid assumption. Generally, Naive+HMM being superior to Greedy+HMM can be expected, since the naive approach for segmentation generates more segments from which the HMM can choose. Naive+HMM outperforming RCP (resp. NLP+RCP) is mostly related to RCP (resp. NLP+RCP) often failing to assign the highest rank to the

correct disambiguation. One important feature of our approach is, as the evaluation confirms, the robustness with regard to the query expression variance. As shown in Figure 3, Naive+HMM achieves the same MRR on natural-language and the keyword-based representation of queries on both – the train and the test – datasets. Overall, Naive+HMM significantly outperforms our baseline Greedy+HNM as well as state-of-the-art techniques based on NLP.

7 Conclusion and Future Work

In this work, we presented an approach to entity disambiguation based on hidden Markov models. Our approach can carry out the parameter estimation necessary to configure the model by employing the knowledge base to query as background knowledge. Our evaluation shows, that our approach achieves significantly better results than state-of-the-art methods based on employing NLP techniques. When excluding queries requiring query expansion (which was out of the scope of this article) we achieve near optimal results.

This work represents a first step in a larger research agenda. Ultimately, we aim to realize a search engine for the Data Web, which is as easy to use as search engines for the Document Web, but allows to create complex queries and returns comprehensive structured query results[9]. A first area of improvements is related to using dictionary knowledge such as hypernyms, hyponyms or co-hyponyms to extend our segments. By these means, we would be able to map *wife* to *spouse*, *daughter* to *child*, etc. Query expansion might also, however, result in a more noisy input for our model. Thus, a careful extension of our approach and analysis of the results will be required. In addition, we will extend our approach with a query cleaning algorithm. The input query might contain some keywords which semantically are not related to the rest of keywords. Since user usually is looking for information semantically closely related to each other, these unrelated keywords (i.e. noise) should be cleaned.

References

1. Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. Technical Report 2003-29, 2003.
2. D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. ACM Press, 2000.
3. D. J. Brenes, D. Gayo-Avello, and R. Garcia. On the fly query entity decomposition using snippets. *CoRR*, abs/1005.5516, 2010.
4. E. Brill and G. Ngai. Man* vs. machine: A case study in base noun phrase learning. ACL, 1999.
5. H. L. Chieu and H. T. Ng. Named entity recognition: A maximum entropy approach using global information. In *Proceedings COLING 2002*, 2002.
6. S.-L. Chuang and L.-F. Chien. Towards automatic generation of query taxonomy: A hierarchical query clustering approach. IEEE Computer Society, 2002.

[9] A prototype of our progress in this regard is available at http://sina.aksw.org.

7. M. Collins and Y. Singer. Unsupervised models for named entity classification. In *SIGDAT Empirical Methods in NLP and Very Large Corpora*, 1999.
8. 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.
9. J. Guo, G. Xu, X. Cheng, and H. Li. Named entity recognition in query. ACM, 2009.
10. T. Joachims, L. A. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR*. ACM, 2005.
11. D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37(2):18–28, 2003.
12. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5), 1999.
13. R. Kraft, C. C. Chang, F. Maghoul, and R. Kumar. Searching with context. In *WWW '06: 15th Int. Conf. on World Wide Web*. ACM, 2006.
14. G. Ladwig and T. Tran. Index structures and top-k join algorithms for native keyword search databases. In *CIKM*. ACM, 2011.
15. S. Lawrence. Context in web search. *IEEE Data Eng. Bull.*, 23(3):25–32, 2000.
16. K. Q. Pu and X. Yu. Keyword query cleaning. *PVLDB*, 1(1):909–920, 2008.
17. Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. *CoRR*, 1995.
18. K. M. Risvik, T. Mikolajewski, and P. Boros. Query segmentation for web search. 2003.
19. S. Shekarpour, S. Auer, A.-C. Ngonga Ngomo, D. Gerber, S. Hellmann, and C. Stadler. Keyword-driven sparql query generation leveraging background knowledge. In *WI-IAT*, 2011.
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. A. Uzuner, B. Katz, and D. Yuret. Word sense disambiguation for information retrieval. AAAI Press / The MIT Press, 1999.
25. E. Vorhees. The trec-8 question answering track report. In *Proceedings of TREC-8*, 1999.
26. J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Query Clustering Using User Logs. *ACM Transactions on Information Systems*, 20(1), 2002.
27. R. W. White, J. M. Jose, C. J. van Rijsbergen, and I. Ruthven. A simulated study of implicit feedback models. volume 2997. Springer, 2004.
28. X. Yu and H. Shi. Query segmentation using conditional random fields. ACM, 2009.
29. Y. Zhu, J. Callan, and J. G. Carbonell. The impact of history length on personalized search. ACM, 2008.