Schema Free Querying of Semantic Data

Lushan Han
University of Maryland,
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland 21250
lushan1@umbc.edu

Tim Finin
University of Maryland,
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland 21250
finin@cs.umbc.edu

Anupam Joshi
University of Maryland,
Baltimore County
1000 Hilltop Circle
Baltimore, Maryland 21250
joshi@cs.umbc.edu

ABSTRACT

While much structured semantic data is available on the Web, we lack intuitive user interfaces enabling casual, non-experts to query it. Natural language query systems are a powerful approach, but current techniques are brittle in addressing the ambiguity and complexity of natural language and require expensive labor to supply the extensive domain knowledge they need. Keyword query systems are relatively easy to use and implement, but are inexpressive and do not support complex queries. We introduce a compromise in which users specify a graphical "skeleton" for a query and annotate it with freely chosen words, phrases and entity names. We describe a framework for interpreting these "schema-free queries" over open domain RDF data that automatically translates them to RDF SPARQL queries. The framework uses statistical approaches to learn domain knowledge, thus avoiding expensive human efforts required by natural language interface systems. We demonstrate the feasibility of the approach with an implementation that performs well in an evaluation on DBpedia data using queries from the 2011 QALD workshop.

1. INTRODUCTION

Developing interfaces to enable casual, non-expert users to query complex structured data has been the subject of much research over the past forty years. Such interfaces allow users to query data without understanding its schema, knowing how to refer to objects, or mastering the appropriate formal query language. A long standing goal has been to allow people to query a database or knowledge-base in natural language, an approach that has seen much work since the 1970s [48, 21, 3, 16, 1]. More recently there has been interest in developing natural language interfaces (NLIs) for XML data [27] and collections of general semantic data encoded in RDF [31, 8, 47, 32, 10].

There are two major obstacles for NLI systems to be widely adopted, however. First, current NLP techniques are still brittle in addressing the ambiguity and complexity of natural language in general [1, 24]. Second, it requires extensive domain knowledge for interpreting natural language questions. Domain knowledge typically consists of a *lexicon*, which maps a user's vocabulary to an ontology vocabulary or logical expressions in NLI systems, and a *world model*, which specifies the relationships between the vocabulary terms (e.g., subclass relationships) and the constraints on the types of arguments of properties. Both can be very expensive when dealing with data in broad/open domains or with heterogeneous schema, such as Semantic Web linked data [5].

Querying structured data with keywords and phrases is an alternative approach that has gained popularity recently [22, 49, 44, 41]. Keyword query systems are more robust than NLI systems because they typically employ a much simpler mapping strategy: map the keywords to the set of elements in the knowledge base that are structurally or associationally close, such as the most specific sub-tree for XML databases [49] and the smallest sub-graph for RDF databases [44]. However, keyword queries have limited expressiveness and inherit ambiguity from the natural language terms used as keywords. For example, the keyword query "president children spouse" can be interpreted either as "give me children and spouses of presidents" or "who are the spouses of the children of presidents". Li et al. [27] compared the performance of a NLI system to a keyword system on a set of complex queries and showed that the keyword system performed poorly against the NLI system.

To precisely query structured data, we must be able to specify the relational structure between the query's key elements. While this can be done in natural language, processing complex, unconstrained sentences is difficult and their potential ambiguity makes choosing the intended interpretation challenging. We introduce a compromise that we call a Schema-Free Query (SFQ) interface, in which users specify a graphical "skeleton" for a query and annotate it with freely chosen words, phrases and entity names. An example is shown in Figure 1. By asking users to specify the semantic relations between entities in a query, we avoid the difficult problem of relation extraction from natural language sentences. While the full expressive power of human language is not supported, people are able to use familiar vocabulary terms in composing a query.

We describe a framework for interpreting SFQs over open domain RDF semantic data and automatically translating them to SPARQL, the standard query language for RDF. Instead of using a manually maintained lexicon, we em-

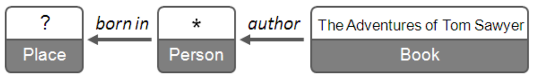

Figure 1: A Schema-Free Query for "Where was the author of the Adventures of Tom Sawyer born?".

ploy a computational semantic similarity/relatedness measure to locate candidate ontology terms for user input terms. Semantic similarity metrics enable our system to have a broader linguistic coverage than that offered by synonym expansion by recognizing non-synonymous terms with very similar meaning. For example, the properties *author of* and *college* are good candidates for the user terms "wrote" and "graduated from", respectively. Semantic similarity measures can be automatically learned from a large domain-dependent corpus.

Knowing the strength of association between concepts and properties is an important kind of domain knowledge that is very useful for disambiguation. The term 'Titanic' in the query "Who are the actors of Titanic" could refer to a ship or a film, but the latter is more likely because films commonly have actors but other potential types (e.g., ship, book, game, place, album, etc.) do not. We know birds can fly but trees cannot, a database table is not kitchen table, etc. Such knowledge is essential for human language understanding. We refer to this as *Concept-level Association Knowledge* (CAK). Domain and range definitions for properties in ontologies, argument constraint definitions of predicates in logic systems and database schemata all belong to this knowledge. However, manually defining this knowledge for broad or open domains is tedious and expensive. In this paper, we introduce an approach to automatically learn this knowledge from semantically marked-up instance data.

With the automatically learned CAK and semantic similarity measures, we present a straightforward but novel algorithm that disambiguates a SFQ and constructs a corresponding SPARQL query to produce an answer. Our algorithm resolves mappings using only concept-level information, i.e., at the schema level. This makes the approach much more scalable than those that directly search into both instance and concept data for possible matches since concept space is much smaller than instance space.

Our initial experiments, briefly described in two poster papers [18, 17], were carried on DBpedia [2], which represents data from Wikipedia as RDF. DBPedia is the key component of the Linked Open Data (LOD) and serves as a microcosm for larger, evolving LOD collections. It provides a broad-based, open domain ontology containing over 300 classes and 1,500 properties currently. Heterogeneity is a problem of the DBpedia ontology because it supplants the categories and attribute names of Wikipedia infoboxes, which were independently designed by different communities. Terms having similar linguistic meanings are used for different contexts. For example, the property *locatedInArea* is for mountains and the property *location* is for companies.

Our approach can be readily applied to any RDF dataset as long as it holds the following properties: (i) class, property and entity names are human-readable words or short phrases; (ii) all relations are binary, (iii) there are no blank nodes or auxiliary nodes; and (iv) only simple value types like *xsd:integer* or *xsd:date* are used. There is already considerable data on the Web meeting these requirements and the volume is growing. Our use of two very general concepts, semantic similarity and association knowledge, enables the approach to be extended to support higher arity relations or complex structures and to be applied to similar graph representations (e.g., Microdata, Freebase, Conceptual Structures).

In the next four sections we present our query interface, describe the automatic learning of concept association knowledge, detail the algorithm for interpreting an SFQ and translating it into SPARQL and present our implementation of semantic similarity measures. An evaluation of our prototype system on test questions from the 2011 QALD workshop is given in Section 6. We discuss related work in Section 7 and conclude our paper by summarizing our contributions and plans for future work in Section 8.

2. SCHEMA-FREE QUERY INTERFACE

A schema-free query (SFQ) is represented as a graph with nodes denoting entities and links representing semantic relations between them. Each entity is described by two unrestricted terms: its name or value and its concept (i.e., class or type). Figure 1 shows an example of a SFQ with three entities (a place, person and book) linked by two relations (*born in* and *author*). Users flag entities they want to see in the results with a '?' and those they do not with a '*'. Terms for concepts can be nouns (*book*) or simple noun phrases (*soccer club*) and relations can be verbs (*wrote*), prepositions (*in*), nouns (*author*), or simple phrases (*born in*). Users are free to reference concepts and relations in their own words as in composing a NL question.

We currently require concept names from users, enabling our system to resolve mappings in concept space rather than instance space. The requirement stems from the observation that people find it easy to explicitly tag the types but it is much harder for machines to infer them. However, we are developing techniques to relax this, as described in the Section 8.

Relation names can be omitted when there is a single "apparent" relation between two concepts that corresponds to the user's intended one. The "apparent" relation, which we call the *default relation*, is typically a *has-relation* or *in-relation*, as shown in the examples in Figure 2. In the first example, a *has-* or *in-relation* exists between *City* and *Country* and in the second, a *has-relation* also exists between *Author* and *Book*. Our system uses a stop word list for filtering relation names with words like *in*, *has*, *from*, *belong*, *of* and *locate*. In this way, a *has-* or *in-relation* is automatically turned into a default relation. The second example in Figure 2 differs from the first in that it can be represented without using a default relation. An author is a person who writes. Since the relation information is implicit in one of the two connected concepts, it need not be explicitly mentioned.

The value of entities can be something other than a name, such as a number or date. If the value of an entity is a number, "Number" is used as the entity's concept. Numeri-

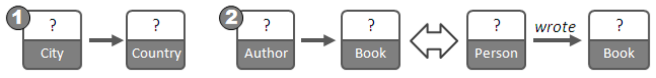

Figure 2: Two examples of default relation.

cal attributes such as population, area, height, and revenue can be thought of as either relations or concepts, but since *Number* is already used as the concept, we require them to be relations. A motivation for the rule is that numerical attributes in the underlying ontology only have simple value types (e.g., *xsd:integer*, *xsd:float*), which we can uniformly treat as *Number* instances. The DBpedia ontology satisfies this constraint.

Like a typical database query language, SFQ can express factual queries but not *why* or *how* questions. We currently support neither numerical restrictions on entity value nor aggregation functions working on the entity in question. We plan to implement these features using form-based fields and drop-lists just below the graph area for creating SFQ. Alternatively, we may implement them as a set of buttons that can be dragged and applied to the entity variables.

By using SFQ interface, we circumvent the yet unsolved problem of *relation extraction* from NL sentences [6, 23, 40, 4]. This is challenging because it has to confront hard linguistic problems such as modifier attachment, anaphora and fine-grained named entity recognition. Extracting relations requires information not only from syntactic level but also from semantic level (e.g., understanding the meaning of the word "same"). Sometimes it also needs common sense knowledge to resolve ambiguity. While modern dependency parsers [29, 11] can achieve about 90% precision and 80% recall, what they generate are grammatical relations between individual words rather than semantic relations between entities. The best systems often rely on machine learning models to extract relations and use dependency parsers to produce features [6, 23], but their performance is still far from reliable.

3. AUTOMATIC CAK LEARNING

We learn Concept-level Association Knowledge statistically from instance data (the "ABOX" of RDF triples) and thus avoid expensive human labor in building the knowledge manually. However, instead of producing "tight" assertions such as those used in RDF property domain and range constraints, we generate the *degree of associations.* Classical logics that make either true or false assertions are less suited in an open-domain scenario, especially those created from heterogeneous data sources. For example, what is the range of the property *author* in DBpedia? Both *Writer* and *Artist* are not appropriate because the object of *author* could be something other than *Writer* or *Artist*, for example *Scientist.* Having *Person* as the range would be too general to be useful. Thus in our case there is no a fixed range for the property *author* but different classes do have varied association strengths of being the type of an object of *author.*

Computing statistical association requires counting the number of occurrences of single terms and co-occurrences of multiple terms in the universe. DBpedia's universe is represented by two datasets: *Ontology Infobox Properties*, which contains RDF triples for all relations between instances, and *Ontology Infobox Types*, which provides all type definitions for the instances.

Figure 3 shows how we count term occurrences and co-occurrences for one relation. On the figure's left is an RDF triple describing a relation and the type definitions for its subject and object and on the right are the resulting occurrences and co-occurrences of terms[1]. We consider direction in counting co-occurrences between classes and properties. The directed co-occurrences are indicated by the arrow character → between two terms, for example *Book→author.* The occurrences of directed classes (e.g. *Book→*) are counted separately from the occurrences of undirected classes (e.g. *Book*).

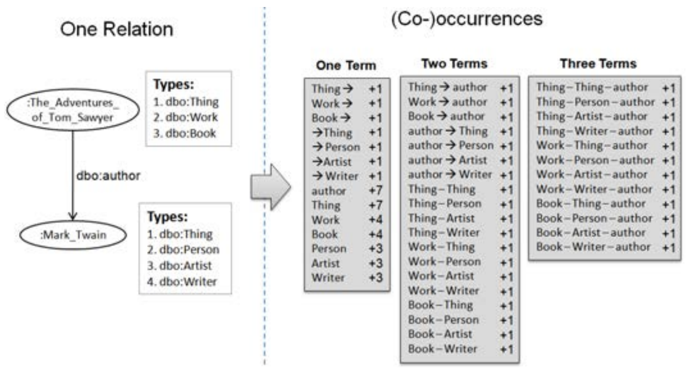

Figure 3: This example shows how we count term occurrences and co-occurrences in an RDF.

Because an instance can have multiple types, the fact that *Mark_Twain* is the object of the property *dbo:author*[2] results in four directed co-occurrences between the property *dbo:author* and each of the types of *Mark_Twain.* Similarly, that *The_Adventures_of_Tom_Sawyer* and *Mark_Twain* are the subject and object of a relation produces twelve pairwise undirected co-occurrences between their types.

After both occurrence and co-occurrence counts are available, we employ the pointwise mutual information (PMI) [7, 19] statistical measure to compute two types of associations: (i) directed association between classes and properties and (ii) undirected association between two classes. Equation 1 gives the PMI formula where f_{t_1} and f_{t_2} are the marginal occurrence counts of the two terms t_1 and t_2 and $f(t_1, t_2)$ is the co-occurrence count of t_1 and t_2 in the universe. N is a constant for the size of the universe.

$$\mathrm{PMI}(t_1, t_2) \approx \log_e\left(\frac{f(t_1, t_2) \cdot N}{f_{t_1} \cdot f_{t_2}}\right) \tag{1}$$

We use the direction-sensitive $\overrightarrow{\mathrm{PMI}}$ to denote the association between a class c and a property p. $\overrightarrow{\mathrm{PMI}}(c, p)$ measures the association degree between c as subject and p as predicate whereas $\overrightarrow{\mathrm{PMI}}(p, c)$ measures the one between p as predicate and c as object. $\overrightarrow{\mathrm{PMI}}$ is computed the same way as PMI except that its class term is directed, as shown below.

$$\overrightarrow{\mathrm{PMI}}(c,\ p) = \mathrm{PMI}(c \rightarrow,\ p) \tag{2}$$

$$\overrightarrow{\mathrm{PMI}}(p,\ c) = \mathrm{PMI}(p,\ \rightarrow c) \tag{3}$$

Our CAK for the DBpedia ontology is stored as two sparse matrices of PMI values between classes and properties and between classes themselves. Figure 4 shows examples of top-25 lists of most associated properties/classes for five terms

[1]Co-occurrences of three terms are maintained for computing conditional probability of properties connecting two given classes, which we will use in the next section.

[2]*dbo* is the RDF namespace prefix for the DBpedia ontology

1) Writer→: @pseudonym 6.0, notableWork 6.0, influencedBy 5.7, skos:subject 5.7, influenced 5.5, movement 5.1, ethnicity 4.3, @birthName 4.3, @deathDate 4.2, relative 4.1, occupation 4.0, @birthDate 3.8, nationality 3.4, education 3.4, child 3.3, award 3.2, deathPlace 3.2, @activeYearsStartYear 3.2, partner 3.2, @activeYearsEndYear 3.1, genre 3.1, spouse 3.0, birthPlace 3.0, citizenship 2.9, foaf:homepage 2.8

2) →Writer: author 6.8, influencedBy 6.4, influenced 6.1, basedOn 5.3, illustrator 5.1, writer 5.1, creator 5.1, coverArtist 4.4, executiveProducer 4.4, relative 4.2, translator 4.1, lyrics 4.0, previousEditor 3.9, editor 3.6, spouse 3.5, child 3.4, nobelLaureates 3.3, designer 3.2, partner 3.2, associateEditor 3.2, director 3.0, narrator 3.0, chiefEditor 2.9, storyEditor 2.8, person 2.7

3) Book→: @isbn 5.8, @numberOfPages 5.8, @oclc 5.6, mediaType 5.6, @lcc 5.6, literaryGenre 5.6, @dcc 5.5, author 5.4, coverArtist 5.2, @publicationDate 5.1, nonFictionSubject 5.1, illustrator 5.1, translator 4.9, publisher 4.9, series 4.5, language 4.0, subsequentWork 3.3, previousWork 3.2, country 1.7, designer -1.9, @meaning -1.9, @formerCallsign -2.1, @review -2.4, @callsignMeaning -2.5, programmeFormat -2.6

4) →Book: notableWork 6.8, firstAppearance 6.4, basedOn 6.1, lastAppearance 5.9, previousWork 5.8, subsequentWork 5.8, series 4.8, knownFor 3.8, notableIdea 3.1, portrayer 2.6, currentProduction 2.3, related 1.9, author 1.7, nonFictionSubject 1.7, writer 1.4, translator 1.1, influencedBy 1.1, significantProject 1.1, award 0.9, coverArtist 0.8, relative 0.5, movement 0.5, associatedMusicalArtist 0.5, associatedBand 0.4, illustrator 0.3

5) author: →Writer 6.8, Musical→ 6.1, Play→ 5.4, Book→ 5.4, Website→ 5.4, WrittenWork→ 5.1, →Journalist 5.0, →Philosopher 4.9, →Website 4.8, →Artist 4.5, →Comedian 4.1, →Person 3.9, →ComicsCreator 3.8, →Scientist 3.6, TelevisionShow→ 3.4, Work→ 3.3, →Senator 3.2, →FictionalCharacter 2.8, →PeriodicalLiterature 2.7, →Governor 2.4, →Wrestler 2.3, →MemberOfParliament 2.3, →OfficeHolder 2.3, →Cleric 2.2, →MilitaryPerson 2.2

Figure 4: Examples of the top-25 most associated properties/classes from DBpedia's CAK

along with their PMI values. Examples 1 to 4 present, in order, outgoing and incoming properties for two classes *Writer* and *Book*. Note that datatype properties are indicated by an initial @ character to distinguish them from object properties. Example 5 shows the classes that could be in domain or range of the property *author*. Terms ending and starting with → are potential domain and range classes, respectively.

In the first four examples, the top properties are the most informative, such as *@pseudonym* and *notableWork* for *Writer* and *@isbn* and *@numberOfPages* for *Book*. Lower ranked properties tend to be less related to the classes. Example two shows that both *author* and *writer* can be incoming properties of *Writer*, though *author* is more related. On the other hand, the third example shows that only *author*, not *writer*, can describe *Book*. In the DBpedia ontology, *author* and *writer* are used for different contexts with *author* used for books. The class *Writer* has both *author* and *writer* as incoming properties because writers can write things other than books (e.g., films, songs). Example five illustrates the heterogeneity of DBpedia's ontology via the property *author*, which carries multiple senses (e.g., book author, Web site creator). Noisy data in DBpedia can result in some abnormal associations, as shown in the fourth example, where *author* can be an incoming property of *Book*. Fortunately, their association strength is typically low.

4. TRANSLATION

We start by laying out the three-step algorithm that maps terms in a SFQ to terms in a target ontology, in this case the DBpedia ontology. The algorithm focuses on vocabulary or schema mapping, which is done without directly involving the instance data. We then discuss how to generate SPARQL queries given the term mappings.

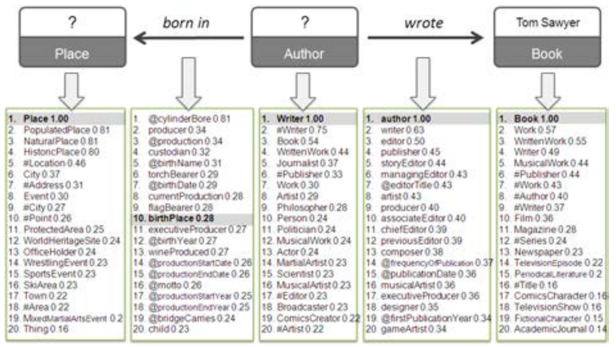

Figure 5: A ranked list of terms from the target ontology is generated for each term in the SFQ, "Who wrote the book Tom Sawyer and where was he born?".

4.1 Mapping Algorithm

4.1.1 Step One: Candidate Generation

For each concept or relation in a SFQ, we generate a list of the k most semantically similar candidate ontology classes or properties. (See Section 5 for semantic similarity computation). A minimum similarity threshold, currently experimentally set at 0.1, guarantees that all the terms have at least some similarity. For a *default relation*, we generate the $\frac{k}{2}$ ontology properties most semantically similar to each of its connected concepts because the semantics of a *default relation* is often conveyed in one of its connected concepts. We also generate $\frac{k}{4}$ ontology properties that are most semantically similar to the words *locate* and *own* on the behalf of "in" and "has", respectively. Finally we assemble these into a list of $\frac{3}{2}k$ ontology properties. The value for k is a compromise between the translation performance and the allowed computation time and depends on the degree of heterogeneity in the underlying ontologies and the fitness of the semantic similarity measure. We currently use an experimentally determined value of 20.

Figure 5 shows the candidate lists generated for the five user terms in the query, with candidates ranked by their similarity score. We use the Stanford part of speech (POS) tagger and morphology package [43] to get word lemmas with their POS and then compute their semantic similarity. While our similarity measure is effective and works well, it is not perfect. For example, "born in" is mistaken as highly similar to "@cylinderBore" and relatively dissimilar to "birthPlace".

Classes starting with # are virtual classes that are automatically derived from the object properties in the target ontology, DBpedia in this case. Many property names are nouns, which can be used to infer the type of the object instance. For example, the object of the *director* property should be a director. Many of these generated types are not included in the native classes but they could nevertheless be entered by users as concepts in a SFQ. Some other examples include *#Chairman*, *#Religion*, and *#Address*. Adding them as auxiliary classes facilitates the mapping.

However, unlike the specifically defined native classes, the virtual classes can be ambiguous. Therefore, we assign them *three fourths* similarity to make them subordinate to native classes.

4.1.2 Step Two: Disambiguation

Each combination of ontology terms, with one term coming from each candidate list, is a potential query interpretation, but some are reasonable and others not. Disambiguation here means choosing the most reasonable interpretations from a set of candidates.

An intuitive measure of reasonableness for a given interpretation is the degree to which its ontology terms associate in the way that their corresponding user terms connect in the SFQ. For example, since "Place" is connected by "born in" in Figure 5, their corresponding ontology terms can be expected to have good association. Therefore, the combination of *Place* and *birthPlace* makes much more sense than that of *Place* and *@cylinderBore* or that of *Place* and *@birthDate* because the CAK tells us that a strong association holds between *Place* and *birthPlace* but not *@cylinderBore* or *@birthDate*. Thus the degree of association from CAK is used as a measure of reasonableness. For another example, CAK data shows that both the combinations of *Writer* and *writer* and of *Writer* and *author* are reasonable interpretations to the SFQ connection "Author → wrote". However, since only *author* not *writer* has a strong association with the class *Book*, the combination of *Writer*, *author* and *Book* produces a much better interpretation than that of *Writer*, *writer* and *Book* for the joint connection "Author → wrote → Book" in the SFQ.

We use two types of connections in a SFQ when computing the overall association of an interpretation: connections between concepts and their relations (e.g., "Author" and "wrote") and between direct connected concepts (e.g., "Author" and "Book"). We exclude indirect connections (e.g., between "Book" and "born in" or between "Book" and "Place") because they do not necessarily entail good associations. This distinguishes from the coarse-grained disambiguation methods [50] where context is a simple a bag of words without compositional structure.

If candidate ontology terms ideally contained all the substitutable terms, we could rely solely on their associations for disambiguation. However, in practice many other related terms are also included and therefore the similarity of the candidate ontology terms to the user's terms is important in identifying the best interpretations. We experimentally found that *weighting their associations by their similarities* produced a better disambiguation algorithm.

To formalize our approach, suppose the query graph G_q has m edges and n nodes. Each concept or relation x_i in G_q has a corresponding set of candidate ontology terms Y_i. Our interpretation space H is the Cartesian product over the sets Y_1, ..., Y_{m+n}.

$$H = Y_1 \times ... \times Y_{m+n} = \{(y_1, ..., y_{m+n}) : y_i \in Y_i\}$$

Each interpretation $h \in H$ also describes a function $h(x)$ that maps x_i to y_i for $i \in \{1, ..., m+n\}$.

Let us define a fitness function $\Phi(h, G)$ that returns the fitness score of an interpretation h on a query graph or subgraph G. We seek the interpretation $h^* \in H$ that maximizes the fitness on the query graph G_q, which is computed as the summation of the fitness on each link L_i in G_q, i from 1 to m. More specifically,

$$h^* = \underset{h \in H}{argmax}\, \Phi(h, G_q) \tag{4}$$

$$\doteq \underset{h \in H}{argmax} \sum_{i=1}^{m} \Phi(h, L_i) \tag{5}$$

where link L_i is a tuple with three elements: subject concept s_i, relation r_i and object concept o_i. Formula 5 achieves *joint disambiguation* because the joint concepts of different links should be mapped to the same ontology class.

Before computing the fitness of link L_i, we first resolve the direction of the ontology property $h(r_i)$ because $h(r_i)$ is semantically similar to r_i but they may have opposite directions. For example, the relation *wrote* in Figure 5 is semantically similar to the property *author* which, however, connects from *Book* to *Author*. Whether the direction of $h(r_i)$ should be inverse to the one of r_i is decided in Formula 6.

$$\begin{aligned}
A\ &= \overrightarrow{\mathrm{PMI}}(h(s_i), h(r_i)) + \overrightarrow{\mathrm{PMI}}(h(r_i), h(o_i)) \\
A' &= \overrightarrow{\mathrm{PMI}}(h(o_i), h(r_i)) + \overrightarrow{\mathrm{PMI}}(h(r_i), h(s_i)) \\
(\hat{s_i},\ \hat{o_i}) &= \begin{cases} (o_i,\ s_i), & \text{if } A' - A > \alpha \\ (s_i,\ o_i), & \text{if } A' - A \leqslant \alpha \end{cases}
\end{aligned} \tag{6}$$

The association terms A and A' measure the degrees of reasonableness for the original and inverse directions, respectively. If the inverse direction is significantly more reasonable than the original, we reverse the direction by switching the classes that $h(r_i)$ connects; otherwise we respect the original direction. Currently, the reverse threshold α is 2.0, based on experimental evidence. The hypothesis behind Formula 6 is that if the two classes are different (e.g., *Author*, *Book*), the properties connecting them tend to go with one direction only (e.g., wrote); if the two classes are the same or similar (e.g., *Actor*, *Person*) their connecting properties can go with both directions (e.g., spouse) but we observed no large differences between the degrees of reasonableness of two directions. Formula 6 works very well empirically. As Section 6 shows, none of incorrect translations of the evaluation queries were caused by mis-resolved directions.

Finally, the fitness on link L_i is the sum of three pairwise associations: the directed association from subject class $h(\hat{s_i})$ to property $h(r_i)$, the directed association from property $h(r_i)$ to object class $h(\hat{o_i})$, and the undirected association between subject class $h(\hat{s_i})$ and object class $h(\hat{o_i})$, all weighted by semantic similarities between ontology terms and their corresponding user terms. More specially,

$$\begin{aligned}
\Phi(h, L_i) = &\overrightarrow{\mathrm{PMI}}(h(\hat{s_i}), h(r_i)) \cdot sim(\hat{s_i}, h(\hat{s_i})) \cdot sim(r_i, h(r_i)) \\
&+ \overrightarrow{\mathrm{PMI}}(h(r_i), h(\hat{o_i})) \cdot sim(\hat{o_i}, h(\hat{o_i})) \cdot sim(r_i, h(r_i)) \\
&+ 2 \cdot \mathrm{PMI}(h(\hat{s_i}), h(\hat{o_i})) \cdot sim(\hat{s_i}, h(\hat{s_i})) \cdot sim(\hat{o_i}, h(\hat{o_i}))
\end{aligned} \tag{7}$$

We use a weight of two for the undirected association term since there are two directed association terms. Moreover, the higher weight for undirected association terms helps in the situations where the corresponding property fails to be in the candidate list of length k. The higher weight gives us a better chance to map the concepts to the corresponding classes via the undirected association term. To facilitate

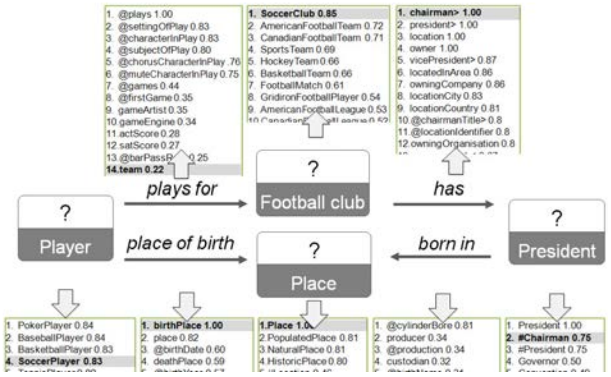

Figure 6: A joint disambiguation example

this, we also impose a lower bound of zero on the two directed association terms to deal with cases where the property $h(r_i)$ fits too poorly with its two classes (their values can be $-\infty$). In these situations the fitness is solely determined by the undirected association term.

Our algorithm can successfully find the correct mappings (marked as bold) for the SFQ in Figures 5. It can also handle more complicated cases such as the one in Figure 6. Some of the mappings are ranked at only 10th and 14th places. The example in Figure 6 is a demonstration of joint disambiguation, which requires taking the context as a whole. The reason *#Chairman* is selected, instead of *President*, is that *President* only means the president of a country in the DBpedia ontology and *SoccerClub* has much higher association with *#Chairman* than with *President*. However, if we take the single link "President → born in → Place" out of the context, *President* will then be preferred over *#Chairman* because almost all presidents are described with their birth places in Wikipedia but not true for "chairmen".

If each candidate list contains k semantically similar terms, the complexity of a straightforward disambiguation algorithm is $O(k^{n+m})$ simply because the total number of interpretations is k^{n+m}. We can significantly reduce this complexity by exploiting locality. The optimal mapping choice of a property can be determined locally when the two classes it links are fixed. So, we only iterate on all k^n combinations of classes. Moreover, we can iterate in a way such that the next combination differs from current combination only on one class with others remaining unchanged. This means we need only re-compute the links involving the changed class. The average number of links in which a class participates is $\frac{2m}{n}$. On the other hand, finding the property that maximizes the fitness of a link requires going through all k choices in the candidate list, resulting in $O(k)$ running time. Put them together, the total computational complexity is reduced to $O(k^n \frac{m}{n} k)$.

Although the running time is still exponential in the number of concepts in G_q, it is not a serious issue in practical applications for three reasons. First, we expect that short queries with a small number of entities will dominate. Second, we can do a much better job in concept mapping than in relation mapping so a small k can be used for producing candidates of concepts and a large k for relations. Finally, we can achieve further improvement by decomposing the graph into subgraphs and/or exploiting parallel computing.

4.1.3 Step Three: Refinement

The best interpretation typically gives us the most appropriate classes and properties for the user terms. For properties, however, two cases that require additional work. The first arises when two connected concepts in G_q are mapped to the correct classes but we are unable to find a reasonable mapping for the relation connecting them. The second occurs when the property being mapped to is an appropriate one but it is not a major property used in the context. Because the two connected concepts are already disambiguated, we use these as the context and consider all of the properties that can connect instances of their corresponding classes.

For a missing property, we map the relation to its most semantically similar property among all connecting properties. In the case of a minor property, our goal is to find the major properties in the context, which may be less similar to the user relation than the minor property but have much higher conditional probabilities. Thus, we use the formula in Equation 8 to identify major properties from all connecting properties. This formula simply trades similarity for popularity. The logarithmic scale is used so that a large difference on popularity can count for only a small difference on similarity. β (currently 0.8) is a coefficient that balances precision and recall.

$$log(\frac{Prob_{major}}{Prob_{minor}}) \cdot \beta > \frac{Sim_{minor}}{Sim_{major}} \qquad (8)$$

4.2 SPARQL Generation

After user terms are disambiguated and mapped to appropriate ontology terms, translating a SFQ to SPARQL is straightforward. Figure 7 shows the SPARQL query produced from the SFQ in Figure 5. Classes are used to type the instances, such as *?x a dbo:Writer*, and properties used to connect instances as in *?0 dbo:author ?x*. The *bif:contains* property is a Virtuoso [12] built-in text search function which finds literals containing specified text. The named entities in the SFQ can be disambiguated by the constraints in the SPARQL query. In this example, *Tom Sawyer* has two constraints: it is in the label of some book and it is written by some writer.

We also generate a *concise* SPARQL query which is produced from the regular one by removing unnecessary class conditions. Removing them compensates for a deficiency in DBpedia: many instances do not have all of the appropriate type assertions. For example, *Bill Clinton* is not asserted to be of type *President*. To address this, we compute the

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?x, ?y WHERE {
  ?0 a dbo:Book .
  ?0 rdfs:label ?label0 .
  ?label0 bif:contains '"Tom Sawyer"' .
  ?x a dbo:Writer .
  ?y a dbo:Place .
  {?0 dbo:author ?x} .
  {?x dbo:birthPlace ?y} .
}
```

Figure 7: This SPARQL query was automatically generated from the SFQ in Figure 5, "Who wrote the book Tom Sawyer and where was he born?".

semantic similarity between properties and classes qualifying the same instance. If they are very similar, we drop the class conditions. For example, in the SPARQL query in Figure 7, *?x* has an incoming property *author* which is semantically similar to its class *Writer*. In this case, we remove the statement *?x a dbo:Writer* because it could be inferred from the property *author*.

5. SEMANTIC SIMILARITY

We need to compute semantic similarity between concepts in the form of noun phrases (e.g., *City* and *Soccer Club*) and between relations in the form of short phrases (e.g., *crosses* and *birth date*). One way is distributional similarity [20], a statistical approach using a term's collective context information drawn from a large text corpus to represent the meaning of the term. Distributional similarity is usually applied to words but it can be generalized to phrases [30]. However, the large number of potential input phrases precludes precomputing and storing distributional similarity data and computing it dynamically as needed would take too long. Thus, we assume the semantics of a phrase is compositional on its component words and apply an algorithm to compute similarity between two phrases using word similarity.

For two given phrases P_1 and P_2, we pair the words in P_1 to the words in P_2 in a way that it maximizes the sum of word similarities of the resulting word-pairs. The maximized sum of word similarities is further normalized by the number of word-pairs. The same process is repeated for the other direction, i.e., from P_2 to P_1. The scores from both directions are then combined using average. The specific metric is shown in Formula 9. Our metric follows the one proposed by Mihalcea [33], but we allow pairing words with different parts-of-speech.

$$sim(P_1, P_2) = \frac{\sum_{w_1 \in \{P_1\}} \max_{w_2 \in \{P_2\}} sim(w_1, w_2)}{2 \cdot |P_1|} + \frac{\sum_{w_2 \in \{P_2\}} \max_{w_1 \in \{P_1\}} sim(w_2, w_1)}{2 \cdot |P_2|} \tag{9}$$

Computing semantic similarity between noun phrases requires additional work. Before running algorithm on two noun phrases, we compute the semantic similarity of their head nouns. If it exceeds an experimentally determined threshold we apply the above metric but with their head nouns being prior-paired and if not, the phrases have similarity of zero. Thus we know that *dog house* is not similar to *house dog*.

Our word similarity measure is based on distributional similarity and latent semantic analysis, which is further enhanced using information from WordNet. Our distributional similarity approach is based on [39], which yields the best performance to date on the TOEFL synonym test [25], with a correctness of 92%. By using a simple context of bag of words, the similarity between words even with different parts of speech can also be computed.

Although distributional similarity has an advantage that it can compute similarity between words that are not strictly synonyms, the human judgments of synonymy found in WordNet are more reliable. Therefore, we give higher similarity to word pairs which are in the same WordNet synset or one of which is the immediate hypernym of the other by adding 0.5 and 0.2 to their distributional similarities, respectively. We also boost similarity between a word and its derivationally related forms by increasing their distributional similarity by 0.3. We do so because a word can often represent the same relation as its derivationally related forms in our context. As examples, "writer" work as the almost same relation to "write" and so does "produce" to "product" because "writer" means the subject that writes and "product" means the thing being produced.

The parameters 0.5, 0.2 and 0.3 were experimentally derived as follows. We first manually generated a set of parameter combinations guided by the intuition that the synonym parameter should be given the highest value and the hypernym and derivational form parameters given values about half as large. We then used a set of test cases from training questions to evaluate each combination of parameters and picked the one yielding the best performance.

In our case, the lexical categories of words are not important; only their semantics matters. However, the value of distributional similarity of words is lowered if they are not in the same lexical category. To counteract this drawback, we put words into the same lexical category using their derivational forms and compute distributional similarity between their aligned forms. Then we compare this value with their original similarity and use the larger one as their similarity.

6. EVALUATION

Dataset. We evaluated our system using a dataset developed for the 2011 Workshop on Question Answering over Linked Data (QALD) [38]. This dataset was designed to evaluate ontology-based question answering (QA) systems and includes 100 natural language (NL) questions (50 training and 50 test) over DBpedia (version 3.6) along with their ground truth answers.

We selected 33 of the 50 test questions (see Table 1) that could be answered using only the DBpedia ontology, i.e., without the additional assertions in the YAGO ontology. Eight of these were slightly modified and their IDs are tagged with a *. Q10, 14, 24, 30, 35, 44 and 45 required modification because they needed operations currently unsupported by our prototype system: aggregation functions (*Which locations have more than two caves?*) and Boolean answers (*Was U.S. President Jackson involved in a war?*). Our changes included removing the unsupported operations or changing the answer type but preserving the relations. For example, the above two questions were changed to *Give me the location of Ape Cave* and *What wars were U.S. President Jackson involved in?*. Although we introduce an auxiliary entity *Ape Cave* for the first question, the entity name does not affect the mapping process since it is done at the schema level and the entity names are not used. In Q37, we substituted "Richard Nixon" for "Bill Clinton" because the original question cannot be answered using the DBpedia ontology only but an entity name change makes it answerable.

Among the 33 questions, six contain two relations (Q2, 3, 29, 35, 37 and 42, marked as italic in Table 1) and the rest only one. In fact, all of the QALD questions have the following patterns that are customized for ontology-based NLI systems: (i) most contain one relation and no more than two; (ii) single answer type or variable; and (iii) no anaphora used. They pose less challenge to NLP parsers but do not fully explore the advantages of graph query.

ID	query	reg., w/o step 3		con., w/o step 3		reg., w/ step 3		con., w/ step 3		time	non-empty	
		prec.	*recall*	*prec.*	*recall*	*prec.*	*recall*	*prec.*	*recall*	(sec.)	*prec.*	*recall*
1	Which companies are in the computer software industry?	1	0.998	1	0.998	1	0.998	1	0.998	2.667	1	0.998
2	*Which telecommunications organizations are located in Belgium?*	0.681	0.852	0.681	0.852	0.681	0.852	0.681	0.852	3.845	0.681	0.852
3	*Give me the official websites of actors of the television show Charmed.*	0.667	0.667	0.667	0.667	1	1	1	1	3.928	1	1
5	What are the official languages of the Philippines?	1	1	1	1	1	1	1	1	1.902	1	1
6	Who is the mayor of New York City?	0	0	0	0	0.125	1	0.125	1	1.730	0.125	1
7	Where did Abraham Lincoln die?	0.667	1	0.556	1	0.667	1	0.556	1	2.101	0.556	1
8	When was the Battle of Gettysburg?	0.667	0.667	0.667	0.667	0.667	0.667	0.667	0.667	1.886	1	1
10*	What is the wife of President Obama called?	0	0	0	0	0	0	0	0	2.311	0.667	0.667
11	What is the area code of Berlin?	0.250	1	0.250	1	0.250	1	0.250	1	2.155	0.250	1
13	In which country is the Limerick Lake?	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	1.994	0.333	0.333
14*	What wars was U.S. President Jackson involved in?	0	0	0	0	0.667	0.389	0.667	0.389	1.637	1	0.583
16	Who is the owner of Universal Studios?	0	0	0	0	0	0	0	0	1.729	0	0
19	What is the currency of the Czech Republic?	1	1	1	1	1	1	1	1	2.247	1	1
24*	What mountains are in Germany?	1	1	1	1	1	1	1	1	2.214	1	1
25	Give me the homepage of Forbes.	0.333	0.333	0.333	0.333	0.333	0.333	0.333	0.333	1.735	0.333	0.333
26	Give me all soccer clubs in Spain.	0	0	0	0	1	1	1	1	2.018	1	1
27	What is the revenue of IBM?	0.250	1	0.250	1	0.250	1	0.250	1	2.069	0.250	1
29	*In which films directed by Garry Marshall was Julia Roberts starring?*	1	1	1	1	1	1	1	1	2.762	1	1
30*	Give me all proteins.	1	1	1	1	1	1	1	1	0.567	1	1
32	Which television shows were created by Walt Disney?	1	0.069	1	0.069	1	0.201	1	0.201	1.716	1	0.201
34	Through which countries does the Yenisei river flow?	0	0	0	0	1	0.500	0.500	0.500	2.022	0.500	0.500
35*	*What city is Egypt's largest city and also its capital?*	0	0	1	1	0	0	1	1	1.887	1	1
37*	*Who is the daughter of Richard Nixon married to?*	1	1	1	1	1	1	1	1	2.464	1	1
40	Who is the author of WikiLeaks?	1	1	1	1	1	1	1	1	2.589	1	1
41	Who designed the Brooklyn Bridge?	0	0	0	0	0	0	0	0	1.734	1	1
42	*Which bridges are of the same type as the Manhattan Bridge?*	0	0	0	0	0	0	0	0	2.099	0	0
43	Which river does the Brooklyn Bridge cross?	1	1	1	1	1	1	1	1	1.644	1	1
44*	Give me the location of Ape Cave.	1	1	1	1	1	1	1	1	1.717	1	1
45*	What is the height of the mountain Annapurna?	0.500	1	0.500	1	0.500	1	0.500	1	1.564	0.500	1
46	What is the highest place of Karakoram?	0.672	1	0.672	1	0.672	1	0.672	1	1.456	0.672	1
47	What did Bruce Carver die from?	1	1	1	1	1	1	1	1	1.721	1	1
49	How tall is Claudia Schiffer?	1	1	1	1	1	1	1	1	1.744	1	1
50	In which country does the Nile start?	0	0	0	0	1	1	1	1	1.693	1	1
Average on 33 queries		**0.546**	**0.604**	**0.573**	**0.634**	**0.671**	**0.736**	**0.683**	**0.766**	**2.047**	**0.754**	**0.832**

Table 1: Average precision, recall and translation time for SPARQL queries generated from 33 questions.

Our system took as input two datasets from DBpedia 3.6: *Ontology Infobox Properties* and *Ontology Infobox Types*. These contain all of the "ABOX" data in the DBpedia ontology. As described in Section 3, we statistically learned Concept-level Association Knowledge from the two datasets and did not use the *DBpedia Ontology* dataset that specifies the class hierarchy and human-crafted domain and range definitions for properties.

Methods and Results. Our system ran on a computer with a 2.33GHz Intel Core2 CPU and 8GB memory. We translated some of the 50 training questions to SFQs and used them to tune our system, including setting various thresholds and coefficients.

Three computer science graduate students who were unfamiliar with DBpedia and its ontology independently translated the 33 test questions into SFQs. We first familiarized the subjects with the SFQ concept and its rules as specified in Section 2 and then trained them with ten questions from the training dataset. We asked them to first identify the entities in a natural language query and their types and then link the entities with the relations given by the query. We also gave them a few simple constraints, e.g., if the entity value is a number, use "Number" as the type of the entity. However, the major force of learning to create the structural queries is by examples. The subjects quickly learned from the ten examples and found the concepts intuitive and easy to understand. The entire learning process took less than half an hour. Finally, we asked each subject to create SFQs for the 33 test questions. Because our graphical web interface was under development, the users drew the queries on paper. None of the subjects had difficulty in constructing the SFQs and all finished within half an hour.

Three versions of the 33 SFQs were given to our system which automatically translated them into four SPARQL queries which are the *regular* and *concise* queries obtained from the best interpretation *with* and *without* step three in the translation process. Table 1 shows the average time to translate a SFQ to the four SPARQL queries, measured in seconds. The queries were then run on public SPARQL endpoints loaded with DBpedia 3.6 to produce answers, which took a few seconds per query. The answers were evaluated for precision and recall, averaging on three versions, as shown in Table 1. The concise queries performed better than regular ones and step three improved performance significantly.

We also evaluated the strategy of issuing multiple queries sequentially until non-empty results are returned. If the concise query generated from the best interpretation with step three gives empty result, we remove the link which has the lowest fitness value and send the modified query again. This process is repeated until no link remains in the query. If no result was obtained, we accepted for the second best interpretation and so on. The performance of this *non-empty* strategy is also shown in Table 1.

Discussion. Relation mapping is more challenging than concept mapping in translating the SFQs to SPARQL because equivalent relations can go beyond synonyms, they can be context-dependent and many of them involve *default relations*. Examples include mapping "actor" to *starring*, "marry" to *spouse*, "die from" to *deathCause*, "mayor" to *leaderName*, “tall” to *height*, “start” to *sourceCountry* and “involved” to *commander*. Thanks to the semantic similarity measure, we are still able to recognize them. Some of them are not similar enough to enter the candidate lists so that they cannot be found at step two. At step three, with context information provided by the disambiguated concepts we then could locate them. For example, in Q50 when we narrow down to the properties occurring between the two classes *River* and *Country*, *sourceCountry* then becomes the most similar to “start”. This explains why the performance of Q6, 14, 26, 34 and 50 was improved by step three.

Structural mismatches between the SFQ and the DBpedia ontology resulted in problems that our current approach has not addressed. We identified two structure mismatch categories: indirect properties and nominal compounds [13].

Wikipedia infoboxes and DBpedia describe the most relevant attributes or relations of concepts, which we call *direct property*. Examples include population, area and the capital of a country, the actors of a film and the maker of a product. Indirect properties are the composition of direct properties. For example, *acted under* between an actor and a director is the composition of two direct properties (*starring* and *director*) joined by a film. As long as the user intentionally uses *direct properties* to compose a SFQ, we expect this kind of structure mismatch would occur infrequently. As for the 33 NL questions, only Q42 contains one *indirect property*.

Some *direct properties* contain duplicate information when one can be inferred by another. For example, in DBpedia there are duplicate *releaseDate* properties associated to both a music album and each song in it. This is unlike normalized relational databases since Wikipedia infoboxes do not preclude redundancy of data. This redundancy also helps alleviate the problem of structural mismatch.

We observed that our users differed in whether a nominal compound should be entered as a phase or decomposed, leading to another category of structure mismatch. For example, two subjects kept the noun phrase “U.S. President” as a single unit while the other decomposed it into two units *President* and *Country* which are linked by the relation *in*. In the DBpedia ontology, however, there are no links between U.S. Presidents and the country United States[3]. Therefore, the SPARQL query translated from the decomposed noun phrase yields an empty result. Q2 and 14 fall in this category. We will present future work dealing with structure mismatch in the last section.

The missing DBpedia class types[4] caused empty results in two queries. In Q10 the entity Obama lacks the type *President* and in Q41 the true answer lacks either *Architect* or *Person* type in the DBpedia Ontology. In their second best interpretations “President” is mapped to the virtual class *#President* and the answer type in Q41 is mapped to *Thing*. Their corresponding SPARQL queries can then produce answers. The missing *City* type for Egypt also resulted in worse performance of regular queries than concise queries in Q35.

The low precision of several queries (Q6, 7, 11, 27 and 45) is caused by entity ambiguity. Q7, for example, might reasonably be interpreted to be about the death of the 16th US president. However, DBpedia includes information on three people with this name, the 16th US president, his grandfather and grandson. Instead of choosing the most notable one, our system generated all. From user’s perspective, it may be best to show a table of all answers along with their URIs and let the user to discriminate herself.

User interpretation of a question can influence its result. In Q7, one subject used the concept *President* for *Abraham Lincoln*, enabling our system to produces the correct answer only. In Q16 all of three subjects interpret “Who” as a *Person* type. However, the type that leads to the correct answer is *Organization*. In Q42 all the subjects decomposed the relation “the same type as” to two relations linking to the same “Type” entity. However, their queries still cannot be translated because the target property, *architecturalBureau*, was not semantically similar to “Type”.

Our disambiguation algorithm sometimes fails due to the flexibility of human expressions. For example, one subject translated Q8 into a “Battle” entity and a “Year” entity which are connected by the relation “took place”. Our system was misled by “took place” because it is much more similar to the property *place* than to *date*. Hence, it mapped “Battle”, “Year” and “took place” to *#Commander*[5], *Event* and inversed *place* respectively, as the best interpretation.

Comparison. The QALD 2011 report [37] showed results of two systems, FREyA and PowerAqua, on the 50 test questions. Both systems modified or reformulated some of the questions that their NLP parsers have difficulties in understanding. We compared our system with them using 30 questions in Table 1. Q24, 44 and 45 were excluded because they had been simplified by removing aggregation operations. Among the 30 questions, FREyA modified four question (Q1, 2, 37 and 50) and PowerAqua eight (Q1, 8, 10, 14, 34, 41, 46 and 50). Average precision and recall of the three systems over the 30 questions is shown in Table 2. We also present their performance on the six questions consisting of two relations. FREyA performs best but it is an interactive system incorporating dialogs to disambiguate questions [10]. This means FREyA sometimes needs users to manually specify the mappings between user terms and ontology terms. PowerAqua’s performance dropped dramatically on the six two-relation questions while FREyA and our system remained the same.

		30 questions		6 two-relations	
		Prec.	*Recall*	*Prec.*	*Recall*
FREyA		0.829	0.849	0.855	0.789
PowerAqua		0.698	0.757	0.167	0.167
Our system	con., w/ step 3	0.668	0.742	0.780	0.809
	non-empty	0.746	0.816	0.780	0.809

Table 2: Comparison on 30 test questions

There are several reasons why our system yields the same performance on six two-relation queries as on other single relation queries. First, we relied on humans to create the relational structure of the queries but PowerAqua uses NLP techniques. Second, two-relation queries give more information and therefore have less ambiguity than single relation queries. The good performance also has something to do with the nature of six two-relation queries. They are fact questions with almost all *direct properties*. However, the

[3]The term “President of United States” appears as the value of a string property of U.S. Presidents, however DBpedia currently does not extract relations from strings

[4]Some of them have been resolved in DBpedia 3.7.

[5]Many *#Commander* instances are countries, resulting in good association between $\rightarrow$*#Commander* and *place*

more relations a query has, the more likely structural mismatches will occur in the mapping. So in general, we would expect performance degrade of our system when working with queries composed of multiple relations but it would still be much better than systems using NLP techniques to understand them.

We also evaluated all 33 test questions on two online systems, PowerAqua [36] and True Knowledge [45] in August 2011. Both include DBpedia as part of their knowledge bases. The true answers of most of the test questions are complete but some are not, which means that PowerAqua and True Knowledge can return correct answers that are not in the true answers of some questions. For these cases, we manually checked the results to identify all correct answers in computing precision. PowerAqua shows the dataset used to derive answers, allowing us to use answers only from DBpedia and ignored others. The results are presented in Table 3.

		33 questions		6 two-relations	
		Prec.	*Recall*	*Prec.*	*Recall*
True Knowledge		0.469	0.535	0.0	0.0
PowerAqua		0.372	0.483	0.168	0.278
Our system	con., w/ step 3	0.683	0.766	0.780	0.809
	non-empty	0.754	0.832	0.780	0.809

Table 3: Comparison to two online systems.

Ontology-based open domain QA is a new research area and the QALD workshop is the first known to us to provide an evaluation dataset. A direct comparison of our system against others is difficult due to different settings. Systems in the comparisons used slightly different query sets and ran on datasets not completely the same. The two online systems have not been tuned using QALD training questions. Moreover, our user interface differs from others. Some people may think either NLI or SFQ interface is just a means to allowing users to describe their information needs and we can directly compare their results. Others may believe the comparison is biased because our system benefits from user interpretation of NL questions.

Nevertheless, the comparisons with top systems show our approach works well. Our system also has three desirable features that others lack. First, our approach saves expensive human effort in crafting schema of data and the mapping lexicon. True Knowledge, FREyA and PowerAqua all depend on such knowledge in performing disambiguation and addressing vocabulary mismatch problem that cannot be solved by synonym expansion [46, 10, 32]. Second, our system has the advantage over automatic NLI systems in answering questions containing two or more relations. It can even handle more complicated queries, such as the ones in Figures 5 and 6, while their corresponding NL questions would inevitably involve multiple answer types and anaphora. Third, our system is fast. FREyA reported 36 seconds on average in answering a question [10]. PowerAqua did not report execution time on QALD questions but our experiment of testing 33 questions on its website showed an average of 143.7 seconds.

7. RELATED WORK

NLIDB systems have been extensively studied since the 1970s [1] and typically take NL sentences as queries and used syntactic, semantic and pragmatic knowledge to produce corresponding SQL queries. Early systems like LUNAR [48] and LADDER [21] were heavily customized to a particular application and difficult to port to other application domains. Later systems, including TEAM [16], ASK [42] and MASQUE [3]. were designed to be portable, allowing knowledge engineers to reconfigure the system before moving to a new domain or letting end users add unknown words through user interaction.

PRECISE [35], a more recent NLIDB system, reduced question interpretation to a maximum bipartite matching problem between the tokens in a NL query and database elements and achieved excellent accuracy on the Mooney dataset without using deep NLP techniques.

Learning semantic parsers are systems that use machine learning methods to map the NL questions into logical forms. These systems, such as SCISSOR [15], have shown very good performance but require manually annotated training data for a specific domain.

A number of portable NLI systems have been developed for ontologies [31, 8, 47, 10] and XML databases [27]. NaLIX [27] translates NL questions to XML queries by mapping the adjacent NL tokens in the parse tree to the neighboring XML elements in the database. ORAKEL [8] constructs a logical lambda-calculus query from a NL question using a recursive computation guided by the question's syntactic structure. FREyA [10] generates a parse tree, maps linguistic terms in the tree to ontology concepts, and formulates a SPARQL query from them with their associated domain and range restrictions. Aqualog [31] and PANTO [47] translating the NL query to linguistic or query triples and then lexically match these to ontology triples. While these systems are portable, they work on one domain at a time.

The last few years have seen a growing interest in open domain NLI systems. True Knowledge [46] and PowerAqua [32] choose pragmatic approaches to turn NL questions into relations. True Knowledge creates 1,200 translation templates to match NL questions. PowerAqua first performs shallow parsing to obtain tokens, POS tags and chunks from NL questions and then use a set of manually-made pattern rules to generate question types and relations. True Knowledge supports user interaction and exploits a repository storing user rephrasing of the questions it cannot understand. PowerAqua extended Aqualog by adding components for merging facts from different ontologies and ranking the results using confidence measures. It runs a potentially expensive graph matching algorithm comparing the query graph to the RDF graph at both data and metadata levels. Treo [14] reduce a NL query to a list of ordered terms guided by the query's dependency structure and matches the terms to RDF paths using semantic similarity measures. It requires recognizing a named entity as the pivot starting a spreading activation process. To our best knowledge, Treo is the only NLI system that also uses semantic similarity to find matchings, but it resolves mapping locally at the instance level and returns ranked triple paths as output.

Substantial research has been done on applying keyword search on structured data, including relational database [22], XML [49, 41] and RDF [44]. Such keyword-based approaches cannot express complex queries and often mix textual content from meta-data and data. A number of approaches [9, 26] extend keyword queries with limited structure information, allowing to specify entity types and attribute-value pairs. However, they are still unable to support querying complex semantics.

A research work closely related to ours is Schema-Free XQuery [28] as it also seeks a middle ground between XQuery (the formal XML query language) and keywords query. The approach allows casual users to create a XQuery without specifying the path expressions between meaningfully related nodes. Instead, a new XQuery operator MLCAS is introduced for "group" the nodes. MLCAS also stands for a XML structure that defines the most specific query context for the related nodes. MLCAS extends the notion LCA (Lowest Common Ancestor) that is commonly used in the XML keyword-based approaches. The Schema-Free XQuery approach focuses on addressing structure mismatch problem while assuming there is no vocabulary mismatch problem or it can simply be solved by synonym expansion. It also requires users understand the XQuery syntax and depends on the LCA structure that is available for trees but not graphs.

8. CONCLUSION AND FUTURE WORK

Large collections of structured semantic data like DBpedia provide essential knowledge for many applications and potentially for end users, but are difficult for non-experts to query and explore. The schema-free structured query approach allows people to query RDF datasets without mastering SPARQL or acquiring detailed knowledge of the classes, properties and individuals in the underlying ontologies and the URIs that denote them. Our system uses statistical data about lexical semantics and the target RDF datasets to generate plausible SPARQL queries from a user's intuitive query. We obtained a promising results in an evaluation on DBpedia with users who sought answers for 33 QALD test questions: precision of 0.754 and recall of 0.832.

A key challenge in our ongoing and future work is to move beyond DBpedia and make it easier to apply our SFQ approach to new RDF data collection and to a large LOD cloud. In addition, we are currently working on three more modest extensions.

The first extension makes entering terms for concepts optional. Consider the SFQ in Figure 5, where the user might omit the concept name for the named entity "Tom Sawyer". Our solution is to find all possible types of entities lexically matching "Tom Sawyer", put the classes into the candidate list of *Tom Sawyer* and run the same algorithm to identify the right class.

The second handles some mismatches between a user's conceptualization of the domain and the target ontology's structure, e.g., a user imagines a *acted under* relation from actors to directors which is absent in the ontology. To support *indirect properties*, we compute a second-order CAK matrix that holds the association degree of concepts two links away. This matrix will be used at step two for measuring undirected association degree between classes. Once the correct classes for the concepts are located, we narrow to their context at step three and find the path matching the *indirect property.* For nominal compounds, we decompose the nouns into two entities linked by a *default relation* and compute the normalized fitness score (divided by the number of links) for the decomposed query, comparing it with the old score to decide if the noun-noun phrase should be broken.

The last incorporates user interaction to give more credibility to answers and improve their accuracy. Instead of directly returning answers we can turn the schema-free query into several "schema-based" queries by replacing terms using the mappings in the top interpretations. Since the user can handle the schema-free query she should be able to understand the "schema-based" queries and choose the most reasonable one or further edit the query. Moreover, information in CAK can be used for creating suggestions that helps users explore concepts in the domain.

9. ACKNOWLEDGEMENT

This research was supported by grants from AFOSR (FA9550-08-1-0265) and NSF (IIS-1250627).

10. REFERENCES

[1] I. Androutsopoulos, G. Ritchie, and P. Thanisch. Natural language interfaces to databases – an introduction. *Natural Language Engineering*, 1(01):29–81, 1995.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *6th Int. Semantic Web Conf.*, pages 722–735. Springer, 2007.

[3] P. Auxerre and R. Inder. Masque modular answering system for queries in english - user's manual. Technical report, Artificial Intelligence Applications Institute, University of Edinburgh, 1986.

[4] M. Banko and O. Etzioni. The tradeoffs between traditional and open relation extraction. In *Proceedings of ACL*, 2008.

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

[6] R. Bunescu and R. Mooney. A shortest path dependency kernel for relation extraction. In *Conf. on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731, 2005.

[7] K. Church and P. Hanks. Word association norms, mutual information and lexicography. In *Proc. 27th Annual Conf. of the ACL*, pages 76–83, 1989.

[8] P. Cimiano, P. Haase, and J. Heizmann. Porting natural language interfaces between domains: an experimental user study with the ORAKEL system. In *Proc. 12th Int. Conf. on Intelligent User Interfaces*, pages 180–189. ACM, 2007.

[9] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSearch: A Semantic Search Engine for XML. In *VLDB*, 2003.

[10] D. Damljanovic, M. Agatonovic, and H. Cunningham. FREyA: An interactive way of querying Linked Data using natural language. In *1st Workshop on Question Answering over Linked Data*, pages 125–138, 2011.

[11] M.-C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *5th Int. Conf. on Language Resources and Evaluation*, pages 449–454, 2006.

[12] O. Erling and I. Mikhailov. RDF support in the virtuoso DBMS. In *Networked Knowledge - Networked Media*, volume 221, pages 7–24. Springer, 2009.

[13] T. Finin. *Semantic Interpretation of Compound Nominals.* PhD thesis, University of Illinois, 1980.

[14] A. Freitas, J. de Oliveira, S. O'Riain, E. Curry, and J. P. da Silva. Querying linked data using semantic

relatedness: A vocabulary independent approach. In *16th Int. Conf. Applications of Natural Language to Information Systems*, pages 40–51. Springer, 2011.

[15] R. Ge and R. J. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proc. of CoNLL-05*, pages 9–16, Ann Arbor, MI, 2005.

[16] B. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: an experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32(2):173–243, 1987.

[17] L. Han, T. Finin, and A. Joshi. GoRelations: An intuitive query system for DBpedia. In *Joint Int. Semantic Technology Conf.* Springer, 2011.

[18] L. Han, T. Finin, and A. Joshi. Schema-free structured querying of DBpedia data. In *21st Conf.on Information and Knowledge Management*, pages 2090–2093. ACM, 2012.

[19] L. Han, T. Finin, P. McNamee, A. Joshi, and Y. Yesha. Improving word similarity by augmenting PMI with estimates of word polysemy. *IEEE Trans. on Knowledge and Data Engineering*, 2012.

[20] Z. Harris. *Mathematical Structures of Language.* Wiley, New York, USA, 1968.

[21] G. Hendrix, E. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *TODS*, 3(2):105–147, 1978.

[22] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[23] N. Kambhatla. Combining lexical, syntactic and semantic features with maximum entropy models. In *Proceedings of ACL*, 2004.

[24] B. Katz and J. Lin. Selectively using relations to improve precision in question answering. In *Proc. of the EACL-2003 Workshop on Natural Language Processing for Question Answering*, 2003.

[25] T. Landauer and S. Dumais. A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge. In *Psychological Review, 104*, pages 211–240, 1997.

[26] Y. Lei, V. Uren, and E. Motta. Semsearch: A search engine for the semantic web. In *15th Int. Conf. on Knowledge Engineering and Knowledge Management*, pages 238–245. Springer, 2006.

[27] Y. Li, H. Yang, and H. Jagadish. Constructing a generic natural language interface for an xml database. In *EDBT*, pages 737–754, 2006.

[28] Y. Li, C. Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB*, pages 72–83, 2004.

[29] D. Lin. Dependency-based evaluation of minipar. In *Workshop on the Evaluation of Parsing Systems*, 1998.

[30] D. Lin and P. Pantel. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):343–360, 2001.

[31] V. Lopez, M. Pasin, and E. Motta. Aqualog: An ontology-portable question answering system for the semantic web. In *Proc. European Semantic Web Conf.*, pages 546–562, 2005.

[32] V. Lopez, V. Uren, M. Sabou, and E. Motta. Cross Ontology Query Answering on the Semantic Web: An Initial Evaluation. In *Proc. 5th Int. Conf. on Knowledge Capture.* ACM, 2009.

[33] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proc. 21st AAAI*, pages 775–780, 2006.

[34] W. Ogden and S. Brooks. Query languages for the casual user: Exploring the middle ground between formal and natural languages. In *SIGCHI Conf. on Human Factors in Computing Systems*, pages 161–165. ACM, 1983.

[35] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proc. 8th Int. Conf. on Intelligent User Interfaces*, pages 149–157. ACM, 2003.

[36] Poweraqua question answering system. http://poweraqua.open.ac.uk:8080/poweraqualinked.

[37] Qald-1 open challenge test phase: Evaluation results. http://bit.ly/QALD11.

[38] 1st workshop on question answering over linked data. http://www.sc.cit-ec.uni-bielefeld.de/qald-1, 2011.

[39] R. Rapp. Word sense discovery based on sense descriptor dissimilarity. In *Proc. 9th Machine Translation Summit*, pages 315–322, 2003.

[40] A. Schutz and P. Buitelaar. Relext: A tool for relation extraction from text in ontology extension. In *Proc. of the 4th ISWC*, pages 593–606, 2005.

[41] A. Termehchy and M. Winslett. Using structural information in xml keyword search effectively. *TODS*, 36(01):4:1–4:39, 2011.

[42] B. Thompson and F. Thompson. Introducing ask, a simple knowledgeable system. In *1st Conf. on Applied Natural Language Processing*, pages 17–24, 1983.

[43] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*, pages 173–180, 2003.

[44] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based Interpretation of Keywords for Semantic Search. In *Proc. of the 6th ISWC*, pages 523–536. Springer, 2007.

[45] Trueknowledge (evi) online system. http://trueknowledge.com/.

[46] W. Tunstall-Pedoe. True knowledge: Open-domain question answering using structured knowledge and inference. *AI Magazine*, 31(3):80–92, 2010.

[47] C. Wang, M. Xiong, Q. Zhou, and Y. Yu. PANTO: A Portable Natural Language Interface to Ontologies. In *Proc. Semantic Web: Research and Applications*, pages 473–487. Springer, 2007.

[48] W. Woods, R. Kaplan, and B. Nash-Webber. The lunar sciences natural language information system. Technical Report 2378, BBN, Cambridge MA, 1972.

[49] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, pages 527–538, 2005.

[50] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd Annual Meeting of the ACL*, pages 189–196, 1995.