UMBC CSEE Technical Report TR-CS-11-08

GoRelations: Towards an Intuitive Query System for RDF Data

Lushan Han and Tim Finin and Anupam Joshi
Computer Science and Electrical Engineering
University of Maryland, Baltimore County
Baltimore, Maryland 21250
{lushan1,finin,joshi}@umbc.edu

01 February 2012

Abstract

Users need better ways to explore DBpedia and obtain information from it. Using SPARQL requires not only mastering its syntax and semantics but also understanding the RDF data model, the ontology used by the DBpedia, and URIs for entities of interest. Natural language question answering systems solve the problem, but these are still subjects of research. We describe a compromise in which non-experts specify a graphical "skeleton" for a query and annotate it with freely chosen words, phrases and entity names. The combination reduces ambiguity and allows us to reliably produce an interpretation that can be translated into SPARQL. We demonstrate the approach's feasibility with an implementation that performs well in an evaluation using queries from the 2011 QALD workshop. Key contributions are the robust methods that combine statistical association and semantic similarity to map user terms to the most appropriate classes and properties used in the DBpedia ontology.

1 Introduction

The growth of Linked Open Data (LOD) has made large amounts of Semantic Web data available. DBpedia [2] is an important example, since it is a key LOD integrating component and can also serve as a microcosm for larger, evolving LOD collections. It provides a broad-based, open domain ontology in which each distinct ontology resource is referenced by a unique URI, a property that stems from Wikipedia's design and the continuous effort of millions of Wikipedians. This facilitates linking data across domains. Other LOD cloud datasets are more domain-specific. Entity co-reference resolution is still indispensable to interlink their data. Most of the datasets are not extensively inter-linked due to the expensive human labor required.

Since Wikipedia infoboxes are designed by different communities and edited by individuals, infobox names and attributes are largely heterogeneous. DBpedia addressed this by manually mapping infoboxes describing the same type of thing to the same DBpedia ontology class and synonymous attributes to the same ontology property, resulting in 320 classes and 1,650 properties currently. Heterogeneity remains a problem for properties due to their large number, lack of domain expertise, and the difficulty of dealing with context-dependent mappings.

For example, the synonymous properties *locatedInArea* and *location* are used in different contexts: one for mountains and the other for companies. There are also synonymous or nearly synonymous properties with the same domain, such as *citizenship* and *nationality*. Properties that are not synonymous can mean the same thing in some contexts. In *a company producing software*, for example, several properties are used: *publisher*, *developer*, *designer*, *product* and *author*. One or two properties often dominate, as do *publisher* and *developer* in our example, which account for more than 85% of the assertions. While DBpedia is a single ontology, it is somewhat similar to the situation where independently developed domain-specific ontologies are combined.

Although SPARQL is available for querying DBpedia, it remains difficult for typical Web users and even experts to query its knowledge base (KB). They must simultaneously master SPARQL, explore the large number of ontology terms, and deal with term heterogeneity. To simplify access, systems like True Knowledge and PowerAqua [13] provide natural language interfaces (NLIs) that allow users to express queries as sentences and and automatically find answers in their underlying KBs. While they are good at answering simple questions (*Who were Richard Nixon's children?* and *Who did Julie Nixon marry?*) they often fail at slightly more complex ones (*Who did President Nixon's children marry?*) due to difficulties in understanding complex natural language (NL) questions [1, 4].

GoRelations (*Graph of Relations*) is an open domain, intuitive query system for DBpedia that is easy to learn and use. It has two components: a *semantic graph interface* (SGI) allowing users to ask queries with complex relations and an effective and efficient automatic translator mapping the semantic graph query into a corresponding SPARQL query to produce an answer. Our current implementation is tailored to DBpedia but the idea is generic and can be extended to other LOD collections. We use semantic similarity metrics to find candidates and statistical association to perform disambiguation. Our approach is adaptable because semantic similarity and statistical association are two general concepts that can be applied to almost all ontologies.

The remainder of the paper proceeds as follows. Section 2 discusses related work on developing simpler RDF query systems. Section 3 describes our Semantic Graph query model. Section 4 details the interpretation of a semantic graph query and its translation into SPARQL. An evaluation of the accuracy of our current implementation on test questions from the 2011 QALD workshop is given in Section 5. We discuss ongoing work on extending the system and conclude the paper in Section 6.

2 Related Work

Work in the 1970s and 1980s focused on developing NLIs for databases [1]. Systems like TEAM [7] took NL sentences as queries, used syntactic, semantic and pragmatic knowledge to produce appropriate SQL queries and applied further pragmatic processing to resolve ambiguity and generate appropriate responses. Such systems required semantic KBs to model the databases and used sophisticated, but somewhat fragile, language processing techniques.

A number of intuitive query systems have been developed for Semantic Web KBs [14, 4, 22, 21, 15, 5], most of which provide NLIs. Trans et al. [21] uses a keyword interface that maps keywords to ontology elements, explores subgraphs where all mapped ontology elements are connected within a maximum distance d, translates each subgraph to a query, and ranks them according to the subgraph's diameter. ORAKEL [4] constructs a logical lambda-calculus query from a NL question using a recursive computation guided by the question's syntactic structure. FREyA [5] generates a parse tree, maps linguistic terms in the tree to ontology concepts, and formulate SPARQL query from them with their associated domain and range restrictions. Aqualog [14], PowerAqua [15], and PANTO [22] translating the NL query to linguistic or query triples and then lexically match these to ontology triples. Our approach does not rely on a NL parser but uses the SGI to get intuitive relations, similar to query triples, directly from users.

All systems need to confront the vocabulary mismatch problem that requires disambiguating user query terms and mapping them to ontology terms. ORAKEL and FREyA acquire mappings through the user interaction and accumulate mappings over time. Aqualog and PANTO provide an automated mapping approach by lexically matching query triples to ontology triples. However, they match the query triples one at a time. Our approach differs by automatically mapping the intuitive relations jointly.

While these systems are portable between different domains, they work only on one or a set of domain-specific ontologies at a time. PowerAqua [15] is an exception as it was designed for working in open-domain scenario and extended Aqualog [14] by adding components for merging facts from different ontologies and ranking the results using confidence measures. GoRelations also targets an open domain scenario, but its focus on DBpedia lessens the problem of cross-ontology entity co-reference resolution. Our mapping approach differs from PowerAqua's in two ways. First, PowerAqua directly searches the ABOX of RDF triples for possible mappings, which can be computationally expensive for large KBs. Our approach resolves mappings using information in the TBOX concept space (classes, properties and their statistics) and thus scales better. Second, PowerAqua relies on lexical matching to find possible mappings whereas we use semantic similarity metrics to find candidates.

3 Semantic Graph Interface

Our interface uses an intuitive notion we call a *semantic graph* (SG) as a representation allowing a user to express a question or description. A semantic graph consists of nodes denoting entities and links representing binary relations between them. Each entity is described by two unrestricted terms: its name or value and its concept in the query context. Figure 1 shows an example of a semantic graph that comprises three entities: a place, person and book, which are 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 '*'.

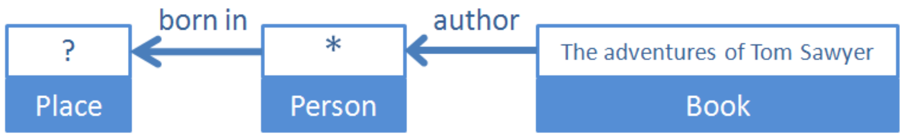


Figure 1: A SG for "Where was the author of the Adventures of Tom Sawyer born?".

Terms for concepts can be nouns (*book*) or simple noun phrases (*soccer club*) and relations can be references as verbs (*wrote*), prepositions (*in*), nouns (*author*) or simple phrases (*born in*). Users are free to name concepts and relations in their own ways as in composing a NL question with a recommendation that concept names be at most two words and relation names three. One reason for the recommendation is that most class and property names in the underlying DBpedia ontology are no longer than two words and three words, respectively. A more fundamental reason is to encourage users to decompose queries into simple entity and relation terms rather than use complex descriptions with internal linguistic structure. For example, we want users to break down *Computer Scientist from US* into the entity *Computer Scientist*, the country *US*, and the relation *from*.

Longer noun phrases are harder to analyze and typically result in a large number of variations that express the same meaning. Allowing two-word noun phrases permits a very productive "qualifier+type" pattern that can be realized in many ways (e.g., *soccer club*, *mountain range*). However, determining whether a two-word noun phrase should be analyzed as a unit or decomposed may not be done consistently by all, leading to unmatched models between the user's query and the underlying ontology. One solution to this problem is to maintain redundancy in the underlying ontology by storing information using both models.

We require concept (i.e., class or type) names, enabling our system to perform disambiguation in concept space rather than instance space. This restriction can be eased by suggesting the most probable concept associated with an entity, as in Query By Example [24]. In contrast, relation names can be left out in a certain circumstance. If there is a single "apparent" relation between two given concepts and it is also the user-intended relation, the user is allowed to leave the relation name empty. 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*. In the second and third examples, default relations also

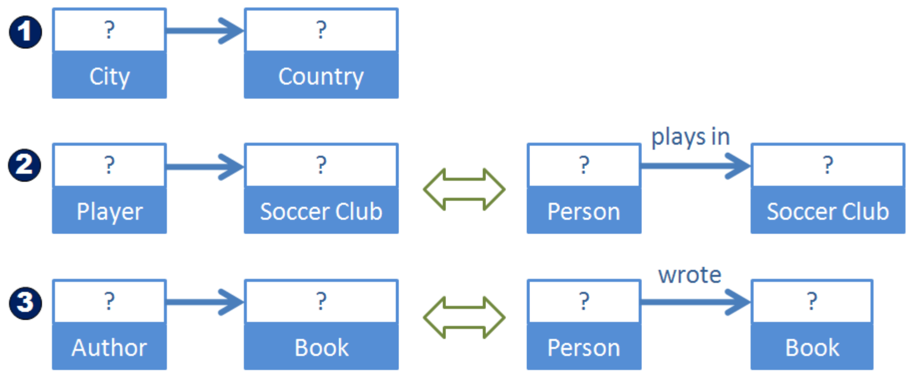


Figure 2: Default relation examples.

exist between *Player* and *Soccer Club* and between *Author* and *Book*. Users can still enter *in* or *has* for the default relation but it is not required. Our system uses a stop word list for filtering relation names that contains words like *in*, *has*, *from*, *belong* and *of*. In this way, a *has-* or *in-relation* is automatically turned into a default relation. Examples two and three are different from the first because they can be represented without using a default relation, as illustrated in Figure 2. A player is a person who plays and an author is a person who writes. Since the information about the relation is already contained in one of the two connected concepts, it need not be explicitly mentioned.

The value of entities can be something other than a name, for example, a number or date. If the value of an entity is a number, "Number" should be used as the entity's concept. Numerical attributes such as population, area, height, and revenue can be thought of as either relations or concepts, but since *Number* is already used as a concept, we require them to be relations. We enforce this rule because in DBpedia's ontology numerical attributes only have data types, which we uniformly treat as *Number* instances.

Using the semantic graph as the user interface has two important advantages when interpreting the user's question. First, since Wikipedia infoboxes capture an entity's most relevant facts as attribute-value pairs, all relations in DBpedia ontology are binary, matching our representation. The use of only binary relations greatly facilitates mapping the user's description of a question to the underlying knowledge base. Second, we circumvent the difficult task of understanding sentential semantics by asking users to directly supply the compositional relations between the lexical terms. On the other hand, users are still left with the freedom of not memorizing any formal language and ontology terms.

By its nature, a semantic graph query is limited in the type of questions it supports. It only allows factual queries but not *why* or *how* questions. Currently, we do not support numerical restrictions on entity value, or aggregation functions working on the entity in question.

4 Translation

We start by laying out the three-step approach that maps terms in the semantic graph to ontology terms. The approach focuses on vocabulary or schema mapping, which is done without involving entities. We then discuss how to generate SPARQL queries from the mappings and address the problem of mapping entities to DBpedia instances or literals. Finally, we describe the ontology statistics and semantic similarity components used in the mapping approach.

4.1 Mapping Approach

To the native classes in DBpedia's ontology we add two data-type classes, *Number* and *Date*, and 230 virtual classes including *#Director*, *#Chairman*, *#Religion*, and *#Address* to help align the user's query model to the underlying ontology model. The added *Number* is intended to subsume all numerical data types (e.g., *xsd:integer* and *xsd:float*) and the *Date* class all temporal data types (e.g., *xsd:date* and *xsd:gYear*). The 230 virtual classes are automatically generated from the object properties in the DBpedia ontology. Many property names are nouns, which can be used to infer the type of the object instance. For example, the object of the property *director* should be a director. However, unlike the specifically defined native classes, the virtual classes can be ambiguous. Many of these generated types are not included in the 320 native classes but they could nevertheless be entered by users as concepts in the semantic graph query. Adding them as auxiliary classes greatly facilitates the mapping task.

Step one: finding semantically similar ontology terms. Like natural language, semantic graphs allow a given question to be expressed in many ways, as shown by the query *Which author wrote The Adventures of Tom Sawyer and where was he born?* in Figure 3. The concept *Author* can be rephrased as *Writer* or *Person* and *Book* can be replaced by *Novel*. The alternatives are either semantically similar (*Author* and *Writer*) or have a subsumption relationship (*Person* and *Writer*) which is typically captured by the ontology's taxonomic structure. As for relations, for example *wrote*, the possible substitutions can be classified into the relations with general meaning, such as *has* and *from*, and the relations with specific meaning including *author of*, *author*, *writer*, *composed*, *book*, *authored*, etc. The relations with general meaning are automatically turned into default relation. The relations with specific meaning, though vary in lexical forms and categories, are semantically similar or related.

The DBpedia ontology does not use general properties with ambiguous names like *has* or *in*. Rather, property names have specific meaning and are naturally described as nouns or noun phrases. Consequently, if relations in a semantic graph also have specific meaning, they are likely to be semantically similar to their corresponding DBpedia property names. For semantic graph default relations, the corresponding ontology properties are often semantically similar to either of the two concepts being related. This is because using one concept's name as relation name is the typical way to represent *has-relation* or *in-relation* in Wikipedia and because the semantics of a default relation is often conveyed in one of its connected concepts.

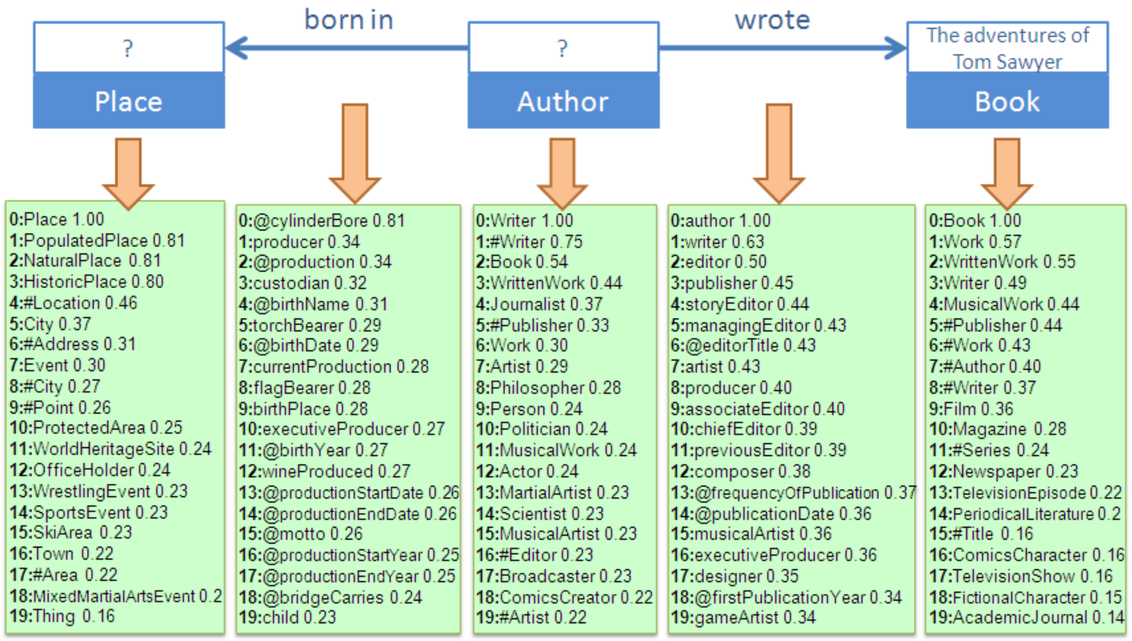


Figure 3: Lists of candidate ontology terms.

For each concept or relation in the semantic graph, we generate a list of the k most semantically similar candidate ontology classes or properties. (See Section 4.4 for semantic similarity computation). A minimum similarity threshold, currently experimentally set at 0.1, is used to guarantee 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. In addition, 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 selection of a 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.

In the example in Figure 3, candidate lists are generated for the five user terms in the semantic graph query. Classes starting with # are virtual classes, which we assign *three fourths* similarity to make them subordinate to native classes. Datatype properties are indicated by a starting @ character to distinguish them from object properties. Candidate terms are ranked by their similarity scores, which are displayed to the right of the terms. Our semantic similarity measure is effective and works well but not perfect. For example, "born in" has mistaken as being highly similar to "@cylinderBore" but has relatively low similarity to "birthPlace". We use the Stanford part of speech tagger and morphology package [20] to get word lemmas and then compute their semantic similarity. This approach works well for most words but fails for "born". The lemma of "born" is "bear", which is has many senses other than "give birth". What is more, the

Stanford POS tagger mistakenly labels the "bore" in "@cylinderBore" as a verb and gives us the wrong lemma, "bear".

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 in this context means choosing the most reasonable interpretations from a large 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 semantic graph.

DBpedia is a knowledge representation of the world's facts made by humans and a semantic graph is a description of some facts about the world in the user's mental model. Since both are mirrors of the world, they share an important feature – associations. Consider the example in Figure 3. In the query graph the relation *wrote* connects the two entities whose concepts under the query context are *Author* and *Book*. This implies that the relation *wrote* should have strong associations with the concepts *Author* and *Book*. What should be reflected in DBpedia's ontology is that the property corresponding to the relation *wrote* should also have good statistical associations with the classes corresponding to the concepts *Author* and *Book*.

Using associations to resolve ambiguity is a common practice, as often seen in word sense disambiguation tasks [16]. Much of the previous work, especially those using unsupervised methods [23, 19], used coarse-grained associations in which a disambiguation context was represented as a bag of words that did not consider the compositional structure of the knowledge encoded by the sentences. This was partly due to the lack of large machine-readable KBs of general knowledge. With DBpedia, which contains tens of millions of facts, we are now able to compute fine-grained associations. We use pointwise mutual information (PMI) [3] to compute pairwise statistical associations between classes and properties and between classes themselves. Basic co-occurrences are counted from all the triples in the DBpedia dataset. When we count the co-occurrences between a class and a property, we also mark whether the class is used to type the subject or the object of the triple. This is important for determining the directions of properties being mapped. See Section 4.3 for details about PMI and how we compute statistical association.

If candidate ontology terms ideally contained all the near-synonyms, we could rely solely on their fine-grained associations for disambiguation. However,in practice many other related terms are also included and therefore the similarity of candidate ontology terms to the user terms is an important feature to identify correct interpretations. We experimentally found that by simply weighting their associations by their similarities we obtain a better disambiguation algorithm.

We present a simple, but novel, disambiguation algorithm that exploits fine-grained associations. Suppose the query graph G has m links and n nodes. We need find a combination of m ontology properties p_1 to p_m, and n ontology classes, c_1 to c_n, from the space H of all interpretations that maximize the goodness or reasonableness of the mapping on the query graph G. This is computed as the summation of goodness of the mapping on each link L_i, i from 1 to m. More specifically,

$$\underset{\mathrm{p_1..p_m\ c_1..c_n \in H}}{\mathrm{argmax}}\ \mathrm{goodness(G)} = \underset{\mathrm{p_1..p_m\ c_1..c_n \in H}}{\mathrm{argmax}} \sum_{\mathrm{i=1}}^{\mathrm{m}} \mathrm{goodness(L_i)} \tag{1}$$

Note that the global optimal mapping on the whole graph is not necessarily composed of all the local optimal mappings on the individual links. Since a node can be involved in multiple links, the mapping decision on the node is affected by all the links it participates in. The local optimal mapping decision from one link may be rejected if it causes low goodness scores on the other links. The same principle can be recursively spread to all other nodes and links in the entire graph. Therefore, our approach maps the semantic graph *jointly.*

To demonstrate the ability of our approach in dealing with more complex queries and doing disambiguation jointly, we introduce a second example in Figure 4. There are four entities in the query graph: a president, player, football club and place. The president and the player are related via two other entities, football club and place. The query means *give me the president of a football club and the players in the club who are born in the same place.* This example is more ambiguous than the one in Figure 3 because the corresponding ontology terms tend to have lower ranks than those in the first one. By taking the context as a whole, the concept *president* should not be mapped to the class *president*, which only stands for the president of a country in DBpedia, although this is the local optimal mapping decision from the link *president born in place.*

Each link L_i is a tuple with three elements: subject concept S_i, relation R_i and object concept O_i. Let their corresponding ontology terms of current interpretation be $c(S_i)$, $p(R_i)$ and $c(O_i)$. Before we compute the goodness of link L_i, we need first resolve the direction of the property $p(R_i)$ because $p(R_i)$ is semantically similar to R_i but they may have opposite directions. For example, the relation *wrote* in Figure 3 is semantically similar to the property *author* which, however, connects from *Book* to *Author.* We invent the statistical association measure $\overrightarrow{\mathrm{PMI}}$ (see Section 4.3) to help determine the direction of $p(R_i)$. $\overrightarrow{\mathrm{PMI}}$ measures statistical association between a class and a property. Unlike the standard PMI, $\overrightarrow{\mathrm{PMI}}$ also considers direction. $\overrightarrow{\mathrm{PMI}}$(Class c, Property p) measures the strength of association between c as subject and p as predicate whereas $\overrightarrow{\mathrm{PMI}}$(Property p, Class c) measures the strength of association between p as predicate and c as object. Whether the direction of $p(R_i)$ should be inverse to the one of R_i is decided in Formula 2.

$$\begin{aligned}
&\text{If } [\overrightarrow{\mathrm{PMI}}(\mathrm{c(O_i), p(R_i)}) + \overrightarrow{\mathrm{PMI}}(\mathrm{p(R_i), c(S_i)})] \\
&\qquad - [\overrightarrow{\mathrm{PMI}}(\mathrm{c(S_i), p(R_i)}) + \overrightarrow{\mathrm{PMI}}(\mathrm{p(R_i), c(O_i)})] > \alpha \\
&\ \text{Then } S_i{}' = O_i,\ O_i{}' = S_i \\
&\ \text{Else } \ S_i{}' = S_i,\ O_i{}' = O_i
\end{aligned} \tag{2}$$

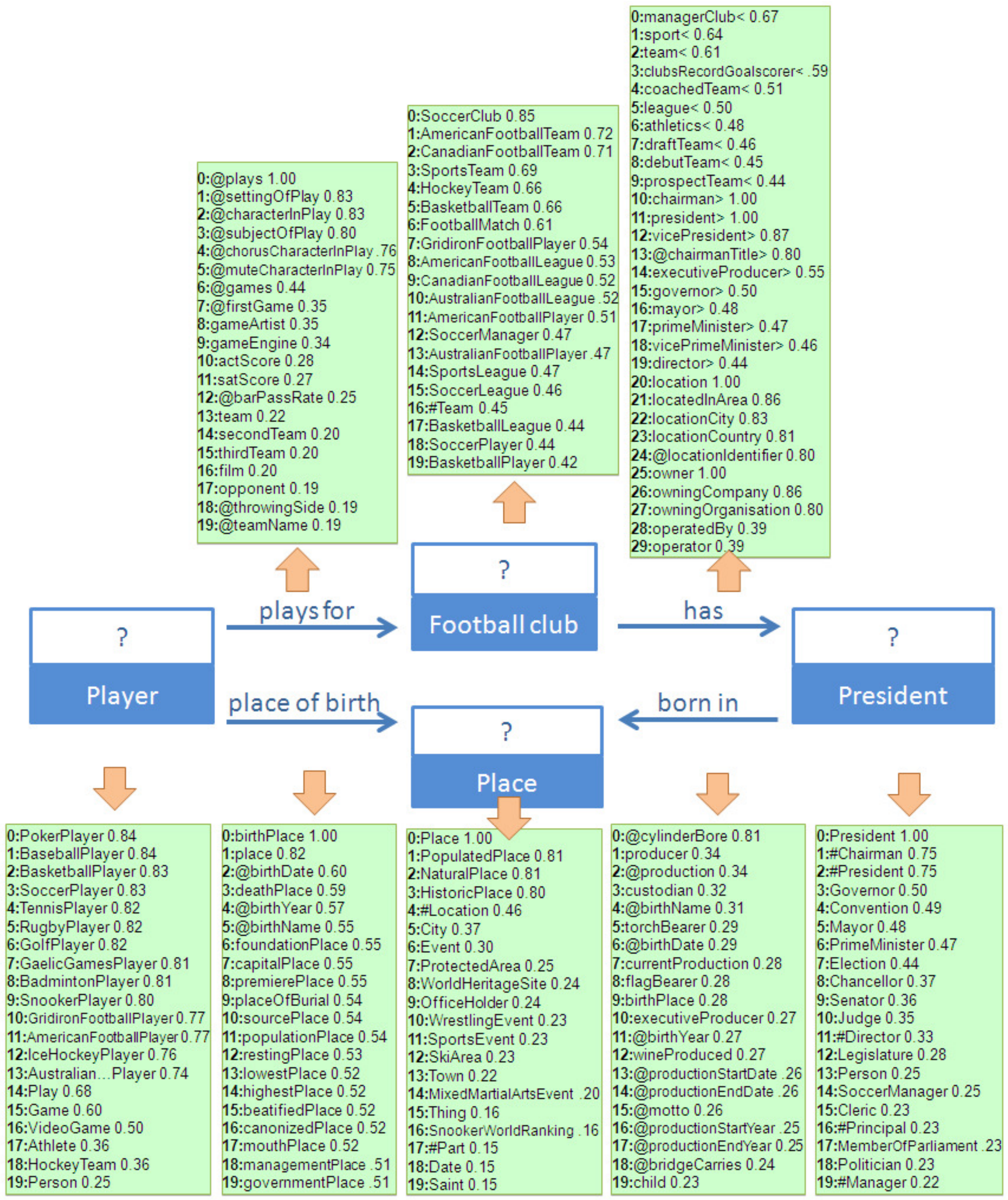


Figure 4: An example demonstrating joint disambiguation.

The association term $\overrightarrow{\mathrm{PMI}}(\mathrm{c(O_i), p(R_i)}) + \overrightarrow{\mathrm{PMI}}(\mathrm{p(R_i), c(S_i)})$ measures the degree of reasonableness of the inverse direction and the term $\overrightarrow{\mathrm{PMI}}(\mathrm{c(S_i), p(R_i)})+\overrightarrow{\mathrm{PMI}}(\mathrm{p(R_i), c(O_i)})$ measures the degree of reasonableness of the original direction. If the inverse direction is much more reasonable than the original direction, we inverse the direction by switching the classes that $p(R_i)$ connects; otherwise we respect the original direction. Currently, the reverse threshold α has been set to 2.0 based on experimental evidence. Setting it within the nearby range, such as 2.5, does not bring visible changes to performance. The hypothesis behind the Formula 2 is that, if the two classes are different (e.g., *Author* and *Book*), in many cases the properties connecting them can go with one direction only (e.g., wrote); if the two classes are the same or similar (e.g., Actor and Person) the properties connecting them can go with both directions (e.g., spouse) but we observed that the degree of reasonableness of two directions typically do not have a big difference. The Formula 2 worked very well empirically. Further verifying the formula using statistical techniques is one of our future work.

Finally, the goodness on link L_i is the sum of three pairwise associations: the directed association from subject class $c(S_i{}')$ to property $p(R_i)$, the directed association from property $p(R_i)$ to object class $c(O_i{}')$, and the undirected association between subject class $c(S_i{}')$ and object class $c(O_i{}')$, all weighted by semantic similarities between ontology terms and their corresponding user terms. More specially,

$$\begin{aligned}
&goodness(L_i) = \\
&\quad \overrightarrow{\mathrm{PMI}}(\mathrm{c(S_i{}'), p(R_i)}) \cdot \mathrm{sim(S_i{}', c(S_i{}'))} \cdot \mathrm{sim(R_i, p(R_i))} \\
&\quad +\overrightarrow{\mathrm{PMI}}(\mathrm{p(R_i), c(O_i{}')}) \cdot \mathrm{sim(O_i{}', c(O_i{}'))} \cdot \mathrm{sim(R_i, p(R_i))}, \\
&\quad +2 \cdot \mathrm{PMI(c(S_i{}'), c(O_i{}'))} \cdot \mathrm{sim(S_i{}', c(S_i{}'))} \cdot \mathrm{sim(O_i{}', c(O_i{}'))}
\end{aligned} \tag{3}$$

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 in which 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 and the other connected links in the graph. To facilitate this, we also impose a lower bound of zero on the two directed association terms to deal with cases where the property $p(R_i)$ fits too poorly with its two classes (their values can be negative infinity). In these situations the goodness is solely determined by the undirected association term.

Of the best interpretation yielded by the disambiguation algorithm for the first example in Figure 3, the concepts *Place*, *Author* and *Book* are mapped to the ontology classes *Place*, *Writer* and *Book* respectively. The relations *born in* and *wrote* are mapped to the ontology properties *birthPlace* and *author* with direction unchanged and reversed respectively. Although the property *birthPlace* has relatively low similarity with *born in*, it is selected because all the candidate terms with higher similarity do not associate well with the classes similar to the concept *Place* and *Author*. The property *writer*, competing for the property *author*, is not selected because it is mainly used for describing films or songs but rarely for books.

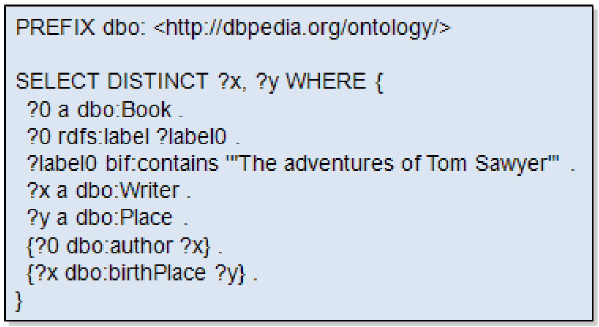
In the best interpretation of the second example, the concepts *President*, *Football Club*, *Player* and *Place* are mapped to *#Chairman*, *Soccer Club*, *Soccer Player* and *Place*, respectively. The *President* is disambiguated to *#Chairman* because *#Chairman* has more association with *Soccer Club* than *President*. Similarly, *Player* is disambiguated to *Soccer Player* because it associates with *Soccer Club* stronger than any other player class. The relation *plays for* is mapped to the corresponding property *team*, which is ranked at only 13th place in the candidate list. The relations *born in* and *place of birth* are both mapped to *birthPlace*, and the default relation *has* is mapped to *chairman*.

Although DBPedia has a certain degree of heterogeneity, it is still a single ontology with fairly good quality. The classes are well defined and form a subsumption hierarchy. In our experiments, we found that we only needed to generate SPARQL query from the best interpretation. However, we expect that in a wider LOD scenario involving multiple datasets and ontologies we will find it necessary to generate SPARQL queries for a set of the most "reasonable" interpretations.

If each candidate list contains k semantically similar terms, the computation 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 can only iterate on all combinations of classes, which have a total number k^n. Moreover, we can iterate in a way such that the next combination differs from current combination only on one class with other classes remain unchanged. This enables us to re-compute only for the links in which the changed class participates and reuse previous computations on other links. The average number of links in which a class participates is $\frac{2m}{n}$. On the other hand, finding the property that maximizes the goodness of a link requires going through all k choices in the candidate list, resulting in $O(k)$ running time. Put them together, the total computation complexity can be reduced to $O(k^n \frac{m}{n} k)$. Further optimization can be achieved by decomposing the graph into subgraphs. We expect that short queries with two or three entities will dominate. For more complex queries, we can decrease k or exploit parallel computing.

Step three: refinement. The best interpretation typically gives us the most appropriate classes and properties for the user terms. However, for properties there are two cases that require additional work. The first arises when the concepts 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 disambiguated property is appropriate but it is not a major property used in the context. Because the two concepts are already disambiguated, we use these as the context and consider all of the properties that can connect instances of their classes along with their conditional probabilities.

In the case of 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 disambiguated property but have much higher conditional probabilities. Thus,



```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?x, ?y WHERE {
 ?0 a dbo:Book .
 ?0 rdfs:label ?label0 .
 ?label0 bif:contains '"The adventures of Tom Sawyer"' .
 ?x a dbo:Writer .
 ?y a dbo:Place .
 {?0 dbo:author ?x} .
 {?x dbo:birthPlace ?y} .
}
```

Figure 5: This SPARQL query was automatically generated from the semantic graph in figure 3.

we use the simple formula in Equation 4 to identify major properties from all connecting properties.

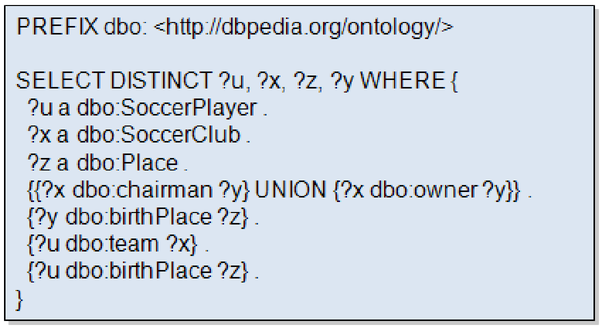
$$log(\frac{Prob_{major}}{Prob_{minor}}) \cdot \beta > \frac{Sim_{minor}}{Sim_{major}} \tag{4}$$

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 is a coefficient which adjusts the balance between precision and recall. We currently set it as 0.8.

Properties in the context sometimes can have an equal or higher similarity to the user relation than the disambiguated property. They tend to be appropriate minor properties. Therefore we also collect them in order to have a better recall. For example, in the best interpretation of the semantic graph in Figure 4, the default relation *has* is disambiguated to the property *chairman* with a similarity 1.0. It turns out that the property *owner* is a minor property in the context (between the classes *#Chairman* and *SoccerClub*) with a similarity 1.0. It is therefore harvested as another appropriate mapping.

4.2 SPARQL Generation

After users terms are disambiguated and mapped to appropriate ontology terms, the translation of a semantic graph query to SPARQL is straightforward. Figure 5 shows the SPARQL query produced from the semantic graph in Figure 3. Classes are used to type the instances, such as *?x a dbo:Writer*, and properties used to connect instances just as relations do for entities as in *?0 dbo:author ?x*. The *bif:contains* property is a Virtuoso built-in text search function which find literals containing specified text. Although we have disambiguated concepts and relations, we have not disambiguated the named entities in the semantic graph, but the SPARQL query can actually do this for us through such constraints. In this example, *The adventures of Tom Sawyer* has two conditions: it is in the label of some book and it is written by some writer.



```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?u, ?x, ?z, ?y WHERE {
 ?u a dbo:SoccerPlayer .
 ?x a dbo:SoccerClub .
 ?z a dbo:Place .
 {{?x dbo:chairman ?y} UNION {?x dbo:owner ?y}} .
 {?y dbo:birthPlace ?z} .
 {?u dbo:team ?x} .
 {?u dbo:birthPlace ?z} .
}
```

Figure 6: This SPARQL query was automatically generated from the semantic graph in figure 4.

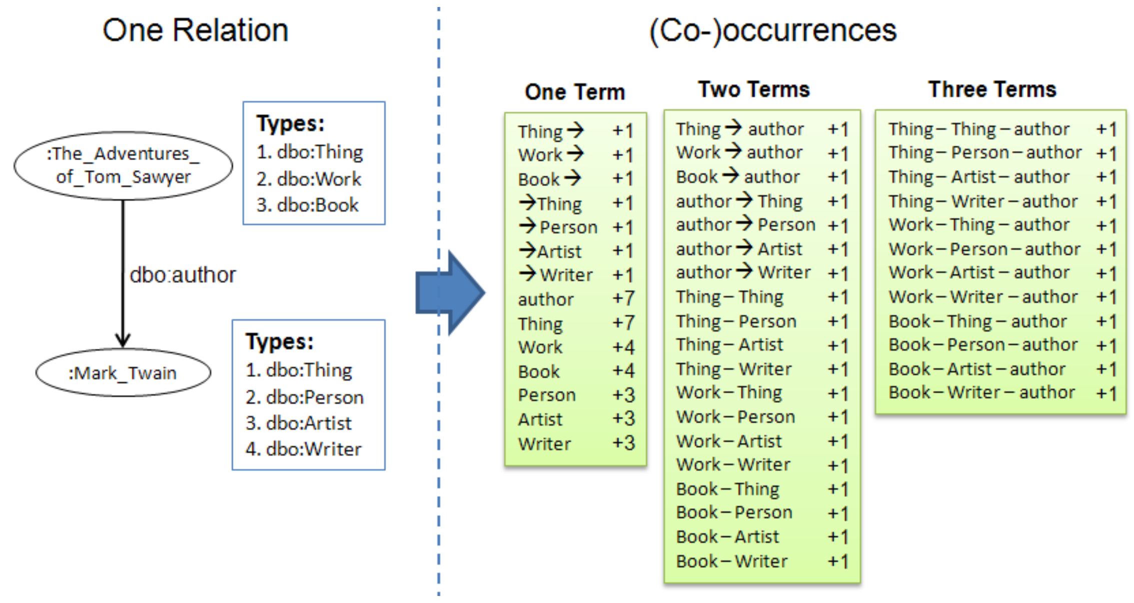
Figure 6 shows the SPARQL query generated for the second example in Figure 4. Since the default relation *has* is mapped to two properties, the SPARQL UNION operator is used to combine them. The assertion that *?y* is an instance of *#Chairman* is already embodied in the triple *?x dbo:chairman ?y*. Although *#Chairman* is an ambiguous class by itself, the conditions in the SPARQL query disambiguate it.

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* and *Beijing* is not of type *City*. To address this, we compute the 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 5, *?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*.

4.3 Ontology Statistics Component

GoRelations uses three kinds of ontology statistics: directed association between classes and properties, undirected association between classes, and conditional probability of properties given two connected classes. Computing these statistics requires information about the number of occurrences of a term and the number of co-occurrences of two or three terms in the universe consisting of all relations. In DBpedia, the universe is represented by the dataset *Ontology Infobox Properties*, which contains RDF triples describing all relations between instances, and the dataset *Ontology Infobox Types*, which provides all type definitions for the instances.

The example in Figure 7 explains how we count term occurrences and co-occurrences by observing one relation in the universe. On the left of the figure, we give an RDF triple describing a relation and the type definitions for the subject and object in the triple. On the right, we list the resulting occurrences and co-occurrences of terms. 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



One Term

Thing→	+1
Work→	+1
Book→	+1
→Thing	+1
→Person	+1
→Artist	+1
→Writer	+1
author	+7
Thing	+7
Work	+4
Book	+4
Person	+3
Artist	+3
Writer	+3

Two Terms

Thing→ author	+1
Work→ author	+1
Book→ author	+1
author→ Thing	+1
author→ Person	+1
author→ Artist	+1
author→ Writer	+1
Thing – Thing	+1
Thing – Person	+1
Thing – Artist	+1
Thing – Writer	+1
Work – Thing	+1
Work – Person	+1
Work – Artist	+1
Work – Writer	+1
Book – Thing	+1
Book – Person	+1
Book – Artist	+1
Book – Writer	+1

Three Terms

Thing – Thing – author	+1
Thing – Person – author	+1
Thing – Artist – author	+1
Thing – Writer – author	+1
Work – Thing – author	+1
Work – Person – author	+1
Work – Artist – author	+1
Work – Writer – author	+1
Book – Thing – author	+1
Book – Person – author	+1
Book – Artist – author	+1
Book – Writer – author	+1

Figure 7: An example for counting (co-)occurrences

separately from the occurrences of undirected classes (e.g. *Book*).

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

PMI [3, 9] is used to measure the strength of statistical association between two terms. Equation 5 gives the basic 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. $\overrightarrow{\text{PMI}}$ is computed the same way as PMI except that its class term is directed. For example, the $\overrightarrow{\text{PMI}}$ value between the property *author* and the directed class →*Person*, 3.93, are significantly smaller than that between *author* and →*Writer*, 6.77, because →*Person* has a much larger number of marginal occurrences than →*Writer* and most occurrences are not associated with *author*.

$$\text{PMI}(t_1, t_2) \approx \log\left(\frac{f(t_1, t_2) \cdot N}{f_{t_1} \cdot f_{t_2}}\right) \tag{5}$$

In a domain-specific ontology, we often use domain and range definitions to qualify properties. However, "tight" domain and range assertions are less suited in an open-domain scenario, especially those created from noisy data sources. For example, what is the range of the property *dbo:author*? Both *dbo:Writer* and *dbo:Artist* are not appropriate because the object of *dbo:author* could be something other than *Writer* or *Artist*, for example *Scientist*. Having *dbo:Person* as the range would be too general to be useful. In real world situations, classical logics that make either true or false assertions are not always applicable. Instead, we often rely on statistical measures that give the probability or degree of correctness. As in our case, there is no a fixed range for the property *dbo:author* but different classes do have varied association strengths of being the object of *dbo:author* statistically.

4.4 Semantic Similarity Component

We need compute semantic similarity between concepts in the form of noun phrases, such as *City* and *Soccer Club*, and between relations in the form of short phrases, such as *crosses* and *birth date*. One way is distributional similarity [10], 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 [12]. 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 that the semantic of a phrase is compositional on its component words and we apply an algorithm to compute semantic similarity between two phrases using word similarity.

As in Mihalcea [17], we pair words from two phrases in a way such 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. 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 run the algorithm 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 human crafted information from WordNet. Our distributional similarity approach, based on [18], yields a correctness of 92% on TOEFL synonym test, which is the best performance to date. 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 a near 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.

In our case, the lexical categories of words are not important and only their semantics matters. However, the value of distributional similarity of words is significantly 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.

DBpedia ontology is a shallow ontology and many subclasses of *Person* class are not included. Consequently, it is possible that some person subtypes appearing in the user query have no similarity to any existing person class in DBpedia ontology. To address this problem, we enforce a lower bound similarity, 0.25, between *person* and any person subtype so that these subtypes can at least be mapped to the DBpedia *Person* class. We use WordNet to find whether a concept in the semantic graph is a person subtype or not.

An ideal semantic similarity measure in our scenario should give high similarity to the terms that can work as synonymous substitution and low similarity to those not. The order of terms with high similarity score is not critical because statistical association can discriminate them and find the most reasonable one. Our implementation has been developed using this strategy. Semantic similarity is an active research field in natural language processing community and has been improved steadily over the years [11, 9]. This component can always benefit from recently progress in this field.

5 Evaluation

To evaluate ontology-based QA systems, the 2011 Workshop on Question Answering over Linked Data provided 50 training and 50 test questions over DBpedia 3.6 along with their ground truth answers. We used the QALD training questions to tune our system, including setting various thresholds and coefficients.

For evaluation, we selected the 33 QALD test questions that could be answered using only the native DBpedia data, i.e., without the additional assertions in the YAGO ontology. Eight of these required slight modifications because they needed operations currently unsupported by our semantic graph notation, such as counts (*Which locations have more than two caves?*), Boolean answers (*Was U.S. President Jackson involved in a war?*) and grouping. Our changes included removing the unsupported operations or changing the answer type but preserving the relations and thus the question schemata. For example, the above two questions were changed to 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. Our collection of 33 test questions with their true answers are available at `http://ebiq.org/r/326`.

Three computer science graduate students who were unfamiliar with DBpedia and its ontology independently translated the test questions into semantic graph queries. We first familiarized the subjects with the semantic graph concept and its rules as specified in Section 3 and then trained them with ten questions from the training dataset. The entire learning process took less than half an hour. Finally, we asked each subject to draw semantic graphs for the 33 test questions. None of the subjects had difficulty in constructing the semantic graph queries and one commented that the experience was somehow like keyword search, except that it was constrained by a structure.

Three versions of 33 semantic graphs 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 *before* and *after* step three in the translation process. The average time to translate a semantic graph to the four SPARQL queries was only two seconds. The queries were then run on public SPARQL endpoints loaded with DBpedia 3.6 to produce answers, which were evaluated for precision, recall and f-measure, averaging on 33 queries of 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 after step-three gives empty result, we remove the link whose three terms have the smallest number of co-occurrences and send the modified query again. This process is repeated until no link remains in the query. If still no result, we go for the second best interpretation and so on. The performance of this strategy is also shown in Table 1. Our prototype system as well as details of the experiments can be accessed online at `http://semanticwebarchive.cs.umbc.edu/GOR/`.

		Prec.	*Recall*	*F*
GoRelations	regular, before step 3	0.546	0.604	0.574
	concise, before step 3	0.573	0.634	0.602
	regular, after step 3	0.671	0.736	0.702
	concise, after step 3	0.683	0.766	0.722
	sequential queries	**0.754**	**0.832**	**0.791**
PowerAqua	1st triple	0.372	0.483	0.420
	all triples	0.334	0.483	0.395
	merged	0.255	0.291	0.272
True Knowledge		0.469	0.535	0.500

Table 1: Precision, recall and f-measure for GoRelations, PowerAqua and True Knowledge on 33 test questions.

The QALD 2011 report identified FREyA [5] as having the best performance. We are unable to directly compare our results with FREyA's for two reasons: it is an interactive system incorporating user feedback and our test question set differs from the one used in QALD as described above.

To allow at least a rough comparison, we evaluated our 33 test questions on two online systems, True Knowledge[1] and PowerAqua[2] in August 2011. Both include DBpedia as part of their knowledge bases, directly or indirectly. The true answers of most of the test questions are complete but some are not. This means True Knowledge and

[1]http://www.trueknowledge.com/

[2]http://poweraqua.open.ac.uk:8080/poweraqualinked

PowerAqua can return correct answers that are not in the true answers of some of the 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. We evaluated PowerAqua in three cases based on its web interface: (i) using top mappings for the first linguistic triple; (ii) using the top mappings for all the linguistic triples; and (iii) merging all of the answers. The results are presented in Table 1. Although GoRelations has higher precision and recall values than the other two systems, it does not necessarily mean that it is better. First, GoRelations has the advantage of getting users' help in interpreting the compositional structure of NL questions while True Knowledge and PowerAqua do not. Second, the tests have not been performed on the exactly same dataset. However, the comparison with these two top systems does show our approach works well.

Our system has a unique feature that it can automatically generate a SPARQL query on DBpedia from the user's description of a question. PowerAqua does not produce SPARQL queries. Some other systems can produce SPARQL queries but they either work on small and close-domain ontologies (e.g., PANTO [22]) or depend heavily on user interactions (e.g., FREyA [5]). Generating a SPARQL query is useful for many reasons: it can be save for reuse, it can be edited to quickly produce variations, and it eventually can support advanced features such as allowing updates to the knowledge-base by non experts. Our approach essentially produces the most appropriate schema in DBpedia for representing the user's information. The semantic graph interface can also work as an input interface for receiving information from end users.

Several questions are themselves ambiguous, preventing our system from producing completely correct answers. The question *Where did Abraham Lincoln die?*, 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. Resolving which entity is intended requires additional information or the application of heuristics, such as preferring the most notable one [6]. What our system chooses is dependent on user interpretation of *Abraham Lincoln*. In the three subject-generated semantic graphs, one used the concept *President* and two used *Person* for the *Abraham Lincoln* entity. Given *President*, our system produces the QALD ground truth answer. Using *Person* it generates two more answers, lowering the precision significantly. Since the NL question often assumes the most popular instance if ambiguity exists, we plan to use a proxy for Google's pageRank to score the ambiguous instances and pick the most popular one to generate its answer. However, from user's perspective, it may be best to show a table of all answers along with the URI of ambiguous instances and let the user to discriminate herself. Another ambiguous question is *Who is the owner of Universal Studios?*. All of three subjects interpret "Who" as a *Person* type. However, the type that leads to the correct answer is *Organization*. Hence, our system failed to ouput the correct translation in all three versions.

The issue of missing class types in DBpedia caused empty results in several cases.

For example, the true answer for the question *Who designed the Brooklyn Bridge?* lacks either *Architect* or *Person* type in the DBpedia Ontology[3]. The SPARQL queries of three versions would all succeed if the correct types were added.

Deciding whether or not to decompose a two-word noun phrase was not done consistently by our subjects, which led to different translations. For example, the noun phrase "U.S. President" was analyzed as a single unit by two subjects 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. Therefore, the SPARQL query translated from the decomposed noun phrase yields an empty result.

Among the 33 questions, six contains two relations and the rest only one. Our system had an even better performance on the six relatively complex questions than the 27 simple ones. This is because the issue of missing class types happens to be less severe on the six questions and two-relation means more context and thus reduces the ambiguity in the questions.

6 Conclusions and future work

GoRelations is an intuitive query system that allows people to query DBpedia without mastering SPARQL or acquiring detailed knowledge of the classes, properties and individuals in the underlying ontologies and the URIs that denote them. It's interface uses a simple *semantic graph* notation for queries that is automatically translated into a corresponding SPARQL query. We developed a novel three-step mapping approach that disambiguates user terms in a semantic graph query and maps them to DBpedia ontology terms. Based on our initial evaluation with several users who sought answers for 33 QALD test questions we obtained a very promising f-measure of 0.791.

While GoRelations currently works on DBpedia, we are working to extend it for an arbitrary collection of LOD resources. The result will be a tool that will allow a broad range of Web users to pose relatively sophisticated queries to a LOD resource and get back useful results.

In the general LOD scenario, we envision a cloud environment where linked data is distributed in multiple SPARQL endpoints while a central server maintains the statistics for the entire data. The central server uses the statistical associations to translate the user's query to the SPARQL queries. These queries are then decomposed and executed on individual SPARQL endpoints. The returned results are joined at the central server to produce the final answer. How to decompose a query using the query information and the statistical information from datasets is a research problem we need to address. We also need consider the linking assertions (e.g., owl:sameAs) in computing statistical associations from LOD collections. The linking assertions will also be used to enable reasoning to integrate results from different SPARQL endpoints.

[3]This problem has been resolved in DBpedia release 3.7

7 Acknowledgments

This research was supported by NSF award IIS-0326460, a gift from Microsoft, and the Human Language Technology Center of Excellence.

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 *Proc. 6th Int. Semantic Web Conf.*, 2007.

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

[4] 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.

[5] D. Damljanovic, M. Agatonovic, and H. Cunningham. Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In *Proc. 7th Extended Semantic Web Conf.* Springer, 2010.

[6] M. Dredze, P. McNamee, D. Rao, A. Gerber, and T. Finin. Entity Disambiguation for Knowledge Base Population. In *Proc. 23rd Int. Conf. on Computational Linguistics*, August 2010.

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

[8] L. Han, T. Finin, and A. Joshi. Gorelations: An intuitive query system for dbpedia. In *Proc. Joint Int. Semantic Technology Conf.*, 2011.

[9] L. Han, T. Finin, P. McNamee, A. Joshi, and Y. Yesha. Improving word similarity by augmenting PMI with estimates of word polysemy. Technical report, Computer Science & Electrical Engineering, U. of Maryland, Baltimore County, June 2011.

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

[11] D. Lin. Automatic retrieval and clustering of similar words. In *Proc. 17th Int. Conf. on Computational Linguistics*, pages 768–774, Montreal, CN, 1998.

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

[13] V. Lopez, M. Fernndez, E. Motta, and N. Stieler. Poweraqua: Supporting users in querying and exploring the semantic web content. *Semantic Web Journal*, 2011.

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

[15] 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.

[16] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, US, 1999.

[17] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. In *Proc. 21st National Conf. on Artificial Intelligence*, pages 775–780, 2006.

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

[19] P. Resnik. Semantic similarity in a taxonomy: An information based measure and its application to problems of ambiguity in natural language. *Journal of Aritificial Intelligence Research*, 11:95–130, 1999.

[20] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. Conf. of the North American Chapter of the Assoc. for Computational Linguistics on Human Language Technology*, pages 173–180, 2003.

[21] T. Tran, P. Cimiano, S. Rudolph, and R. Studer. Ontology-based Interpretation of Keywords for Semantic Search. In *Proc. 6th Int. Semantic Web Conf.*, 2007.

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

[23] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. 33rd annual meeting of the Assoc. for Computational Linguistics*, pages 189–196, 1995.

[24] M. Zloof. Query by example. In *Proc. National Computer Conf. & Exposition*. ACM, 1975.