Natural Language Interfaces to Ontologies: Combining Syntactic Analysis and Ontology-Based Lookup through the User Interaction

Danica Damljanovic, Milan Agatonovic, and Hamish Cunningham

Department of Computer Science, University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, UK
{d.damljanovic,m.agatonovic,h.cunningham}@dcs.shef.ac.uk

Abstract. With large datasets such as Linked Open Data available, there is a need for more user-friendly interfaces which will bring the advantages of these data closer to the casual users. Several recent studies have shown user preference to Natural Language Interfaces (NLIs) in comparison to others. Although many NLIs to ontologies have been developed, those that have reasonable performance are domain-specific and tend to require customisation for each new domain which, from a developer's perspective, makes them expensive to maintain. We present our system FREyA, which combines syntactic parsing with the knowledge encoded in ontologies in order to reduce the customisation effort. If the system fails to automatically derive an answer, it will generate clarification dialogs for the user. The user's selections are saved and used for training the system in order to improve its performance over time. FREyA is evaluated using Mooney Geoquery dataset with very high precision and recall.

Keywords: Natural language interfaces, ontologies, question-answering, learning, clarification dialogs.

1 Introduction

With billions of triples being published in recent years, such as those from Linked Open Data[1], there is a need for more user-friendly interfaces which will bring the advantages of these data closer to the casual users. Research has been very active in developing various interfaces for accessing structured knowledge, from faceted search, where knowledge is grouped and represented through taxonomies, to menu-guided and form-based interfaces such as those offered by KIM [16]. While hiding the complexity of underlying query languages such as SPARQL[2], these interfaces still require that the user is familiarised with the queried knowledge

[1] http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

[2] http://www.w3.org/TR/rdf-sparql-query/

L. Aroyo et al. (Eds.): ESWC 2010, Part I, LNCS 6088, pp. 106–120, 2010.

structure. However, casual users should be able to access the data despite their queries not matching exactly the queried data structures [8]. Natural Language Interfaces (NLIs), which are often referred as closed-domain Question Answering (QA) systems, have a very important role as they are intuitive for the end users and preferred to keyword-based, menu-based or graphical interfaces [9].

Most QA systems contain the classifier module which is used to detect the question category or the type of the question. The successful parsing is based on this identification. However, the syntactic patterns for this classification are usually derived from the dataset which must be large in order to work efficiently [7]. Moreover, as Ferret et al. point out: "answers to [some] questions can hardly be reduced to a pattern." [7, p.7]. In addition, it is not trivial to translate successfully parsed question into the relevant logical representation or a formal query which will lead to the correct answer [14].

We present an approach where instead of encoding the specific rules into our NLI system (which would increase the chance of being domain-specific), we use the knowledge encoded in ontologies as the primary source for understanding the user's question, and only then try to use the output of the syntactic parsing in order to provide the more precise answer. This allows more freedom and flexibility to the user, as questions do not need to fall within the predefined categories. If the system is not able to automatically derive an answer, it will generate clarification dialogs. The user's choice is then saved and used for *training* the system in order to improve its performance over time. While engaging the user in this kind of interaction might be an overload at the beginning, our intention is to see whether our learning mechanism can reduce this by the time. On the route to achieving this, we have developed FREyA – a system which combines several usability methods and is named after **F**eedback, **R**efinement and **E**xtended Vocabular**y** **A**ggregation. We evaluate our approach using Mooney Geoquery dataset[3].

This paper is structured as follows. In the next section we present related work. In Section 3, we describe FREyA and demonstrate through examples how it works. In Section 4 we present evaluation results with Mooney dataset. Finally, we conclude and draw future directions in Section 5.

2 Related Work

While NLI systems which have a good performance require a customisation (such as in the case of ORAKEL [1]), several systems have been developed for which the customisation is not mandatory (e.g., PANTO [19], Querix [10], AquaLog [13]), QuestIO [5], NLP-Reduce [10]). However, as is reported in [13] the customisation usually improves the recall. On the other hand, some of these systems rely on grammatically correct questions which fall within the boundaries of system capabilities.

In case of ORAKEL, customisation is performed through the user interaction, using a software called FrameMapper, where the linguistic argument structures,

[3] `http://www.cs.utexas.edu/users/ml/nldata.html`

such as verbs or nouns with their arguments, are mapped to the relations in the ontology. While their intention is to involve application developers in this customisation, our system is intended to be used by end-users from the start. We are aware that initial system given to the end-users can be seen as overload as the users will be heavily engaged into the dialogs, until the system *learns* enough to be able to automatically suggests the correct answer.

AquaLog [13] is capable of learning the user's jargon in order to improve his experience by the time. Their learning mechanism is good in a way that it uses ontology reasoning to learn more generic patterns, which could then be reused for the questions with similar context. Our approach is different in that it shares the input from all users and reuses it for the others, not favorising jargon of the particular user.

Querix [11] is another ontology-based question answering system which relies on clarification dialogs in case of ambiguities. Our system is similar to Querix in many aspects, with the main difference that the primary goal of the dialog in our system is not only to resolve ambiguities, but also to map question terms to the relevant ontology concepts. Therefore, our system does not rely on the vocabulary of the ontology, but tries to align it with that of the user.

Our intention with FREyA is to balance between heavy customisation which is usually required by application developers, in order to port the system to a different domain, and the end users who need to explore the available knowledge without being constrained with the query language.

3 FREyA

In the previous work ([5], [18]), we have developed QuestIO (Question-based Interface to Ontologies), which translates a Natural Language (NL) or a keyword-based question into SPARQL, and returns the answer to the user after executing the formal query against an ontology. Although this approach uses very shallow NLP, it is quite efficient for very small and domain-specific ontologies. Also, it performs quite well for the set of ill-formed and grammatically incorrect questions [5]. However, the trade-off is that many grammatically correct questions which do require more deep analysis would remain unanswered, or partially answered. For example, if the question is *What is the largest city in Nevada?*, QuestIO would be able to list cities in Nevada, but it would ignore the word *largest* which is in this case crucial to deliver semantic meaning. In addition, when ontologies are spanning diverse domains, automatically deriving an answer becomes an issue due to ambiguities. Finally, QuestIO displays the result of executing SPARQL queries as a table in which the user finds the answer. Therefore, we have started to work on methods which would, in comparison to our previous work:

- improve understanding of the question's semantic meaning
- provide the concise answer to the user's question
- communicate the system's interpretation of the query to the user
- assist the user formulate the query which falls within the boundaries of the system capabilities

These methods have been thoroughly discussed in [3]. Their combination is the base of FREyA which is named after **F**eedback, **R**efinement and **E**xtended Vocabular**y** **A**ggregation. The implementation of FREyA can be broken down into several steps:

- Identification and verification of *ontology concepts*
- Generating SPARQL
- Identification of the *answer type* and presenting the results to the user

3.1 Identification and Verification of Ontology Concepts

Our algorithm for translating a NL question into the set of Ontology Concepts (OCs)[4] combines the syntactic parsing with ontology reasoning in order to identify the user's information need correctly. In cases when the algorithm does not derive conclusions automatically, it generates suggestions for the user. By engaging the user into the dialog, we have a better chance of identifying his information need when it is expressed ambiguously through the question.

Figure 1 shows step-by-step process which starts with finding ontology-based annotations in the query and ends with a set of ontology concepts, which are then used in subsequent steps to generate a SPARQL query. Further we describe each step in more details.

Identification of Ontology Concepts. We use the knowledge available in the ontology to identify the ontology-based annotations in the question, which we call Ontology Concepts. Generated annotations contain links to ontology resources (e.g. URIs). If there are ambiguous annotations in the query, we engage the user into the dialog. For example, if someone is enquiring about *Mississippi*, we might not be able to automatically derive whether OC refers to *geo:River*[5], or *geo:State*. To resolve this ambiguity, we generate a clarification dialog where the user selects one of the two. Note that we apply disambiguation rules which are based on the ontology reasoning before we model the clarification dialog. For example, for the question *which rivers flow through Mississippi?*, modeling a clarification dialog is not necessary, as due to the context of the question we automatically derive that *Mississippi* refers to *geo:State*.

We use GATE [2] application, and an ontology-based gazetteer called OntoRoot Gazetteer [5] to perform this step. OntoRoot Gazetteer relies on the human understandable lexicalisations of ontology resources and therefore, the quality of produced annotations depends directly on them (see [5]). However, it is not often the case that ontology resources are followed by human understandable lexicalisations (e.g., labels). This is especially the case for properties. In addition, Natural Language is so complex that words like *total*, *smallest*, *higher*

[4] Note that we use the term *Ontology Concept* to refer to all types of ontology resources such as classes, instances, properties and literals.

[5] For clarity of presentation, we use prefix *geo:* instead of `http://www.mooney.net/geo#`in all examples.

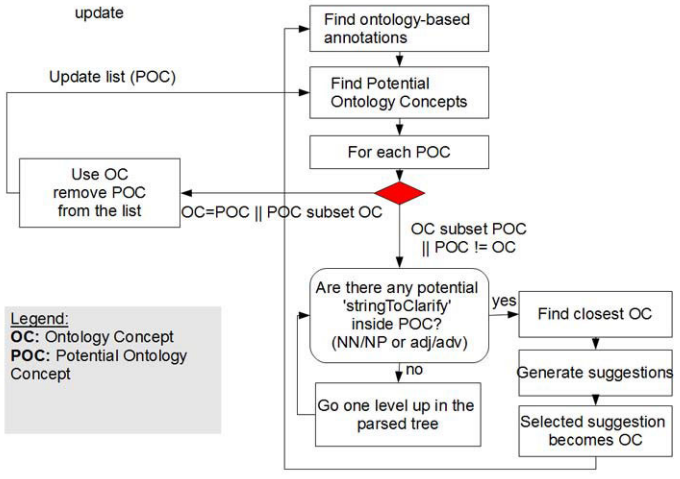

Fig. 1. Validation of potential ontology concepts through the user interaction

than or *how many* cannot be automatically understood/encoded into the relevant structure without additional processing. Some formal languages such as SPARQL even do not support some of these structures (e.g., it is not possible to do count queries in SPARQL). Therefore, NLIs to ontologies would have to translate any additional semantic meanings into the relevant operations with the ontology concepts (e.g. superlative means applying maximum or minimum function to the datatype property value).

Identification of Potential Ontology Concepts. *Potential Ontology Concepts* (POCs) are derived from the syntactic parse tree, and refer to question terms which could be linked to an ontology concept. Syntactic parse tree is generated by Stanford Parser [12]. We use several heuristic rules in order to identify POCs. For example, each NP (noun phrase) or NN (noun) is identified as a POC. Also, if a noun phrase contains adjectives, these are considered POCs as well. Next, the algorithm iterates through the list of POCs, attempting to map them to OCs.

Mapping POCs to OCs. A Potential Ontology Concept is mapped to Ontology Concept in two ways:

1. *Automatically*: if it overlaps with an Ontology Concept in a way that OC spans over POC:
   - both POC and OC refer to exactly the same text span in the question ($OC == POC$); for example, in *which rivers flow through Texas?*, *rivers* can be identified as OC, as referring to the class *geo:River*, while it can also be identified as POC. In this case, POC is automatically mapped to OC, as $OC == POC$ (the starting and ending offsets are identical)

   - POC refers to the text span which is contained within the span to which OC refers ($POC \subset OC$);
2. *By engaging the user*: when the user verifies it by choosing it from the list of the available suggestions

Generating Suggestions. Suggestions are generated for each POC which does not overlap with OC, or in cases when POC spans over OC ($OC \subset POC || POC! = OC$). First, our algorithm identifies the *closest OC* to this POC by walking through the syntax tree, and then uses ontology reasoning to generate suggestions. Based on the type of the closest OC, rules for generating suggestions vary (see Table 1).

Table 1. Generating suggestions based on the type of the closest OC

Type of the closest OC	**Generating suggestions**
class or instance	*get all classes connected to OC by exactly one property, and all properties defined for this OC*
datatype property of type number	*maximum, minimum and sum function of OC*
object property	*get all domain and range classes for OC*

Option *none* is always added to the list of suggestions (see Table 2). This allows the user to ignore suggestions, if they are irrelevant. That is, the system would assume that the POC in the dialog should not be mapped to any suggested OCs, and therefore the system would learn by the time that this POC is either: 1) incorrectly identified, or 2) cannot be mapped to any OC as the ontology does not contain relevant knowledge. While this option will not be of a huge benefit to the end-users, it is intended to identify flaws in the system and encourage improvements.

The task of creating and ranking the suggestions before showing them to the user is quite complex, and this complexity arises as the queried knowledge source grows.

Ranking Suggestions. Initial ranking is based on the string similarity between POC and suggestions, and also based on synonym detection as identified by Wordnet [6] and Cyc[6]. For string similarity we combine Monge Elkan[7] metrics with Soundex[8] algorithm. When comparing the two strings the former gives a very high score to those which are exact parts of the other. For example, if we compare *population* with *city population*, the similarity would be maximised as the former is contained in the latter. The intuition behind this is the way ontology concepts are usually named. Soundex algorithm compensates for any spelling mistakes that the user makes - this algorithm gives a very high similarity to the two words which are spelled differently but would be pronounced similarly.

[6] `http://sw.opencyc.org/`
[7] See `http://www.dcs.shef.ac.uk/~sam/stringmetrics.html#monge`
[8] `http://en.wikipedia/wiki/Soundex`

Table 2. Sample queries and generated suggestions for relevant POCs

Query	POC	Closest OC	Suggestions
population of cities in california	population	geo:City	1. city population 2. state 3. has city 4. is city of 5. none
population of california	population	geo:california	1. state population 2. state pop density 3. has low point ... n. none
which city has the largest population in california	largest population	geo:City	1. max(city population) 2. min(city population) 3. sum(city population) 4. none

3.2 Generating SPARQL

After all POCs are resolved, the query is interpreted as a set of OCs. Firstly, we insert any potential *joker* elements in between OCs, if necessary. For example, if the first two OCs derived from a question are referring to *a property* and *a class* respectively, one *joker* class would be added before them. For instance, the query *what is the highest point of the state bordering Mississippi?* would be translated into the list of the following OCs:

```
geo:isHighestPointOf  geo:State  geo:border  geo:mississippi
PROPERTY              CLASS      PROPERTY    INSTANCE
```

These elements are transformed into the following:

```
?      geo:isHighestPointOf geo:State geo:border geo:mississippi
JOKER  PROPERTY1            CLASS1    PROPERTY2  INSTANCE
```

Next step is *generating set of triples from OCs*, taking into account the domain and range of the properties. For example, from the previous list, two triples would be generated[9]:

```
? - geo:isHighestPointOf - geo:State;
geo:State - geo:borders - geo:mississippi (geo:State);
```

Last step is *generating SPARQL query*. Set of triples are combined and based on the OC type, relevant parts are added to *SELECT* and *WHERE* clauses. Following the previous example, the SPARQL query would look like:

[9] Note that if *geo:isHighestPointOf* would have *geo:State* as a domain, the triple would look like:*geo:State - geo:isHighestPointOf - ?;*.

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix geo: <http://www.mooney.net/geo#>
select ?firstJoker  ?p0  ?c1  ?p2  ?i3
where { { ?firstJoker ?p0 ?c1 .
filter (?p0=geo:isHighestPointOf) . }
?c1 rdf:type geo:State .
?c1  ?p2 ?i3 .
filter (?p2=geo:borders) .
?i3 rdf:type geo:State .
filter (?i3=geo:mississippi) . }
```

3.3 Answer Type Identification

The result of the SPARQL query is a graph, and an important decision to make is how to display results to the user. In order to show the concise answer, we must identify the *answer type* of the question. To achieve this, we combine the output of the syntactic parsing with the ontology-based lookup coupled with several heuristic rules (see [4] for detailed algorithm). Figure 2 shows how we display the answer for the query *Show lakes in Minnesota*.

Fig. 2. List showing the answer of the query *Show lakes in Minnesota*

In addition, as feedback can help the user familiarise himself with the queried knowledge structure, we also render *the system's interpretation of the query*: this is visualised as a graph, where we place the answer type in the center, and the answer on the nearest circle, see Figure 3. We use JIT library[10] for graph visualisation.

3.4 Learning

We use an approach inspired by Reinforcement Learning (RL) [17] to improve ranking of suggestions shown to the user. While many question-answering systems apply supervised learning, we decide to use semi-supervised approach due to several reasons. Firstly, supervised learning goes in-line with automatic classification of the question, where each question is identified as belonging to the

[10] www.thejit.org

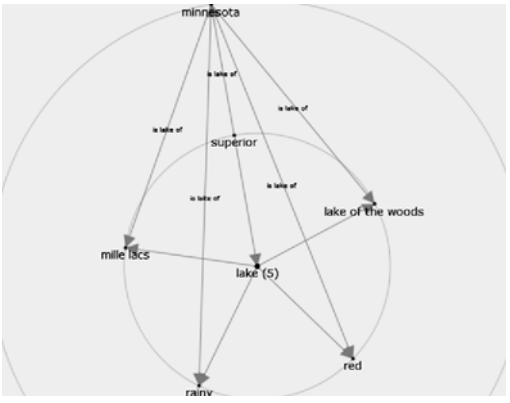

Fig. 3. Graph showing the system interpretation of the query *Show lakes in Minnesota*

one predefined category. Our intention with FREyA is to avoid automatic classification and allow the user to enter queries of any form. Secondly, we want to minimise customisation of the NLI system which would be required when using supervised learning, as to map some question terms to the underlying structure. For example, we want the system itself to suggest that *where* should be mapped to the specific ontology concept such as *Location*, rather than the application developer browsing the ontology structure in order to place this mapping.

The first important aspect of RL is the identification of the goal to be achieved, which is in our case the correct ranking of suggestions. Each suggestion has its *initial ranking* calculated based on synonym detection and string similarity as explained previously. These are used in the untrained system. Each time the suggestion is selected by the user, it receives a reward of +1 while all alternative ones receive -1. The system then learns to place the correct suggestion at the top for any *similar* questions. *Similar* is identified by a combination of a POC and the closest OC. This increases robustness of our learning mechanism as our learning model is not updated per question, but per each combination of POC and the closest OC. In addition, we apply some generalisation rules derived from the ontology. For example, if the closest OC is *geo:Capital*, we would save its superclass *geo:City* in our learning model in order to reuse the same rule for all *cities*, not only *capitals*.

An example which demonstrate how the learning algorithm works, is shown in Figure 4. For query *What is the highest point of the state with the largest area?* there is only one token (*state*) annotated as referring to OC, whereas there are three POCs. We start with the last POC *largest area*. Suggestions are generated based on the closest OC which is *geo:State*. As *geo:stateArea* is a datatype property of type number, generated suggestions would, among others, contain the following: *max(geo:stateArea*, *min(geo:stateArea)*, *sum(geo:stateArea)*. If the user selects the first one, the system will *learn* that the *largest area* refers to the maximum value of *geo:stateArea*.

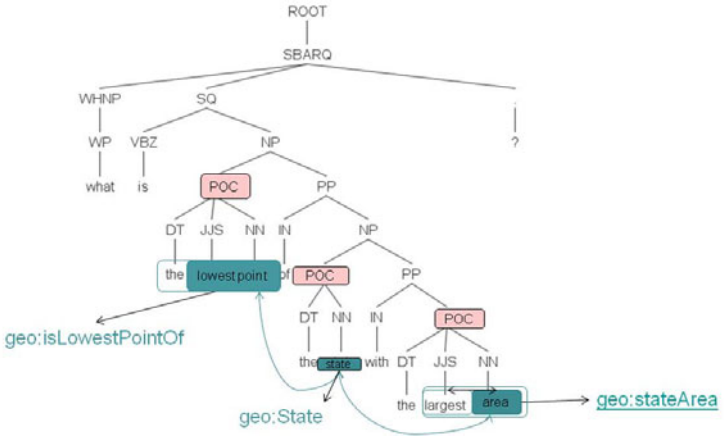

Fig. 4. Mapping POCs to OCs

We then skip the next POC as it overlaps with the ontology concept *state*. The last POC *the lowest point* is then used to generate suggestions. The closest OC is again, *geo:State*. There will be several suggestions and the user is very likely so select *geo:isLowestPointOf*, although this one will be ranked third[11]. However, for the next user, the system will learn to rank *geo:isLowestPointOf* first.

Figure 5 shows the syntax tree for the query *what is the population of New York*. As *New York* is identified as it could be referring to both *geo:State* and *geo:City*, we first ask the user to disambiguate (see Figure 5 a.)). If he selects for example *geo:City*, we start iterating through the list of POCs. The first POC (*New York* as *geo:City*) overlaps with already identified OC, which causes its immediate verification so we skip it. The next one (*population*) is used, together with the closest OC *geo:City*, to generate suggestions. Among them there will be *geo:cityPopulation* and after the user select this from the list of available options, *population* is mapped to the datatype property *geo:cityPopulation* (see Figure 5 b.)). Note that if the user selected that *New York* refers to *geo:State*, suggestions would be different, and following his selection, *population* would probably be mapped to refer to *geo:statePopulation* as the closest OC would be *geo:State*.

4 Evaluation

We have evaluated our approach on the 250 questions from the Mooney Geoquery dataset. Although the ontology contains rather small portion of knowledge about

[11] Note that although *lowest* is a superlative it will not be further used to generate suggestions as *geo:isLowestPointOf* is an object property, while in case of *largest area*, one of the options was *max(geo:stateArea)*, as *geo:stateArea* is datatype property of type number.

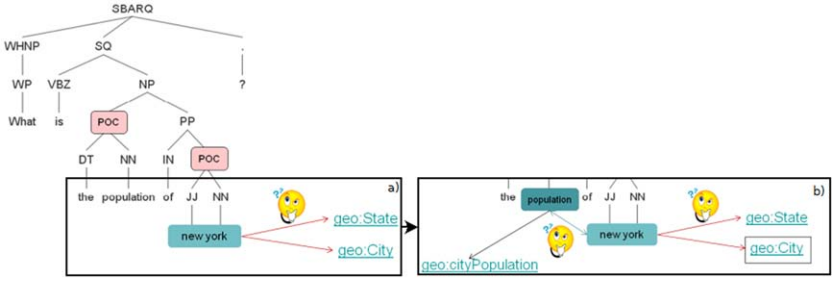

Fig. 5. Validation of potential ontology concepts through the user interaction

the geography of the United States, the questions are quite complex and the system must have a good understanding of the semantic meaning in order to correctly answer them. We evaluate *correctness*, *ranked suggestions*, and *learning mechanism*.

4.1 Correctness

We report correctness of FREyA in terms of precision and recall, which are measures adapted from information retrieval (see [15],[14]). *Recall* is defined as the number of questions correctly answered by an NLI system, divided by the total number of questions in the dataset. *Precision* measures the number of questions correctly answered divided by the number of questions for which the answer is returned at all [1].

Table 3 shows the number of questions correctly answered automatically, as opposed to those which have been answered correctly only after engaging the user into at most 2 clarification dialogs. Finally, there is a system failure to answer questions correctly in 7.6% of the time (e.g., questions with negation). Recall and precision values are equal, reaching 92.4%. This is due to FREyA always returning an answer, although partial or incorrect.

Table 3. Results of running FREyA with 250 questions from Mooney Geoquery dataset

Correct			Incorrect
no dialogs	**1 dialog**	**2 dialogs**	
72	127	32	19
28.8%	50.8%	12.8%	7.6%

If we neglect the fact that it required quite an engagement of the user in order to correctly answer questions, FREyA's performance favorably compares to other similar systems. PANTO [19] is a similar system which was evaluated with the Mooney geography dataset of 877 questions (they removed duplicates from the original set of 879). They reported precision and recall of 88.05% and

85.86% respectively. NLP-Reduce [10] was evaluated with the original dataset, reporting 70.7% precision and 76.4% recall. Kaufmann et al.[11] selected 215 questions which syntactically represent the original set of 879 queries. They reported the evaluation results over this subset for their system Querix with 86.08% precision and 87.11% recall. Our 250 questions are a superset of these 215. It should be noted that FREyA had quite poor performance in comparison to others if we consider automatically answered questions only. However, the intention with FREyA is to allow users full control over the querying process.

4.2 Ranked Suggestions

We use *Mean Reciprocal Rank (MRR)* to report the performance of our ranking algorithm. MRR is a statistic for evaluating any process that produces a list of possible responses (suggestions in our case) to a query, ordered by probability of correctness. The *reciprocal rank* of a suggestion is the multiplicative inverse of the correct rank. The *mean reciprocal rank* is the average of the reciprocal ranks of results for a sample of queries (see Equation 1).

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{Q} \frac{1}{rank_i} \tag{1}$$

We have manually labeled the correct ranking for suggestions which have be generated when running FREyA with 250 questions. This was the gold standard against which our ranking mechanism achieved MRR of 0.81. However, for some cases it is very hard to judge automatically which suggestion to place as number one. It is very likely that different users would select different suggestions for the questions phrased the same way. This emphasises the importance of the dialogs when modeling NLI systems.

4.3 Learning

From the above set of 250 questions, we have randomly selected 103 which required one clarification dialog with the user in order to get the correct answer. Then, we have ran our initial ranking algorithm and compared results with manually labeled gold standard. MRR was 0.72. Table 4 shows the distribution of the rankings.

Table 4. Evaluation with 103 questions from Mooney geography dataset

Correct rank	**Number of questions**
1	64 (62.13%)
2 or 3	22 (21.36%)
4 or more	17 (16.5%)

We then grouped 103 questions by OC, and then randomly chose training and evaluation sets from each group. We repeated this two times. Table 5 shows the structure of the dataset grouped by OC for both iterations. Note that these two iterations are independent - they have both been performed starting with an untrained system.

Table 5. Distribution of the training and evaluation datasets for 103 questions

	Iteration 1		**Iteration 2**	
OC	**Training**	**Evaluation**	**Training**	**Evaluation**
geo:State	26	19	19	26
geo:City/Capital	20	19	19	20
geo:River	12	6	9	9
geo:Mountain	1	0	0	1
total	59	44	47	56

After learning the model with 59 questions from the iteration 1, MRR for the evaluation questions (44 of them) has reached 0.98. Overall MRR (for all 103 questions) increased from 0.72 to 0.77. After training the model with 47 questions during the iteration 2, overall MRR increased to 0.79. Average MRR after running these two experiments was 0.78, which shows the increase of 0.06 in comparison to MRR of the initial rankings. Therefore, we conclude that for the selection of 103 questions from the Mooney Geoquery dataset, our learning algorithm improved our initial ranking by 6%.

5 Conclusion and Future Work

We presented FREyA, a NLI to ontologies which balances between heavy customisation (which is usually required by application developers, in order to port the NLI system to a different domain), and the end users who need to explore the available knowledge without being constrained with the query language. FREyA combines the output of the syntactic parser with the ontology-based lookup in order to approve the user's information need and, if necessary, engage the user into the dialog. Our evaluation with Mooney Geoquery dataset, shows that FREyA compares favorably to other similar systems. Moreover, this system is *learning* from the user's clicks in order to improve its performance over time. Our evaluation with 103 questions from the Mooney dataset revealed that the learning algorithm improved the initial rankings of suggestions by 6%.

At present, we are experimenting with large datasets such as LDSR[12], which have been developed as a part of LarKC project[13]. LDSR includes several subgraphs of Linked Open Data. We are preparing a user-centric evaluation in order to measure the user's experience with FREyA.

[12] `http://ldsr.ontotext.com`

[13] `http://www.larkc.eu`

Acknowledgments

We would like to thank Abraham Bernstein and Esther Kaufmann from the University of Zurich, for sharing with us Mooney dataset in OWL format, and J. Mooney from University of Texas for making this dataset publicly available.

This research has been partially supported by the EU-funded MUSING (FP6-027097) and LarKC (FP7-215535) projects.

References

1. Cimiano, P., Haase, P., Heizmann, J.: Porting natural language interfaces between domains: an experimental user study with the orakel system. In: IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces, pp. 180–189. ACM, New York (2007)
2. Cunningham, H.: Information Extraction, Automatic. Encyclopedia of Language and Linguistics, 2nd edn., pp. 665–677 (2005)
3. Damljanovic, D., Bontcheva, K.: Towards enhanced usability of natural language interfaces to knowledge bases. In: Devedzic, V., Gasevic, D. (eds.) Special issue on Semantic Web and Web 2.0, vol. 6, pp. 105–133. Springer, Berlin (2009)
4. Damljanovic, D., Agatonovic, M., Cunningham, H.: Identification of the Question Focus: Combining Syntactic Analysis and Ontology-based Lookup through the User Interaction. In: 7th Language Resources and Evaluation Conference (LREC). ELRA, La Valletta (May 2010)
5. Damljanovic, D., Tablan, V., Bontcheva, K.: A text-based query interface to owl ontologies. In: 6th Language Resources and Evaluation Conference (LREC). ELRA, Marrakech (May 2008)
6. Fellbaum, C. (ed.): WordNet - An Electronic Lexical Database. MIT Press, Cambridge (1998)
7. Ferret, O., Grau, B., Hurault-plantet, M., Illouz, G., Monceaux, L., Robba, I., Vilnat, A.: Finding an answer based on the recognition of the question focus (2001)
8. Hurtado, C.A., Poulovassilis, A., Wood, P.T.: Ranking approximate answers to semantic web queries. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E.P.B. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 263–277. Springer, Heidelberg (2009)
9. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519. Springer, Heidelberg (2007)
10. Kaufmann, E., Bernstein, A., Fischer, L.: NLP-Reduce: A naive but domain-independent natural language interface for querying ontologies. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, Springer, Heidelberg (2007)
11. Kaufmann, E., Bernstein, A., Zumstein, R.: Querix: A natural language interface to query ontologies based on clarification dialogs. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 980–981. Springer, Heidelberg (2006)
12. Klein, D., Manning, C.D.: Fast exact inference with a factored model for natural language parsing. In: Becker, S., Thrun, S., Obermayer, K. (eds.) Advances in Neural Information Processing Systems 15 - Neural Information Processing Systems, NIPS 2002, pp. 3–10. MIT Press, Cambridge (2002), `http://books.nips.cc/papers/files/nips15/CS01.pdf`

13. Lopez, V., Uren, V., Motta, E., Pasin, M.: Aqualog: An ontology-driven question answering system for organizational semantic intranets. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 72–105 (2007)
14. Mooney, R.J.: Using multiple clause constructors in inductive logic programming for semantic parsing. In: Proceedings of the 12th European Conference on Machine Learning, pp. 466–477 (2001)
15. Popescu, A.M., Etzioni, O., Kautz, H.: Towards a theory of natural language interfaces to databases. In: IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces, pp. 149–157. ACM, New York (2003)
16. Popov, B., Kiryakov, A., Kirilov, A., Manov, D., Ognyanoff, D., Goranov, M.: KIM – Semantic Annotation Platform. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 834–849. Springer, Heidelberg (2003)
17. Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction. MIT Press, Cambridge (1998)
18. Tablan, V., Damljanovic, D., Bontcheva, K.: A natural language query interface to structured information. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 361–375. Springer, Heidelberg (2008)
19. Wang, C., Xiong, M., Zhou, Q., Yu, Y.: Panto: A portable natural language interface to ontologies. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 473–487. Springer, Heidelberg (2007)