QAnswer - Enhanced Entity Matching for Question Answering over Linked Data

Stefan Ruseti[1,2], Alexandru Mirea[1], Traian Rebedea[1,2], and Stefan Trausan-Matu[1]

[1] University Politehnica of Bucharest, Romania
[2] TeamNet International, Bucharest, Romania
{stefan.ruseti, traian.rebedea, stefan.trausan}@cs.pub.ro
alexandru.daniel.mirea@gmail.com

Abstract. QAnswer is a question answering system that uses DBpedia as a knowledge base and converts natural language questions into a SPARQL query. In order to improve the match between entities and relations and natural language text, we make use of Wikipedia to extract lexicalizations of the DBpedia entities and then match them with the question. These entities are validated on the ontology, while missing ones can be inferred. The proposed system was tested in the QALD-5 challenge and it obtained a F1 score of 0.30, which placed QAnswer in the second position in the challenge, despite the fact that the system used only a small subset of the properties in DBpedia, due to the long extraction process.

1 Introduction

Question answering systems have been considered an important goal of Artificial Intelligence and a subject of research for many years. While the traditional way of searching for information is most of the times based on keywords, a much more natural way of searching for information is using natural language questions. Also, instead of generating a list of documents where the answer can be found, such a system would return a precise answer.

Usually, question answering systems either use a knowledge base with structured information, or try to extract the answer from free text, or employ a combination of these two approaches. Knowledge bases contain more precise information, but cannot cover all the possible questions. In order for a system to answer a question, it must first be able to "understand" the meaning of the question. This part usually consists of translating the natural language in a form of structured information, which is used later for querying a knowledge base.

Our system uses the structured knowledge base DBpedia and a Wikipedia-based approach to match phrases from the question to entities in the ontology. Different solutions for matching each type of entity were developed and the most probable interpretation is converted in a SPARQL query. Missing properties or types can also be inferred if they were not matched in the previous step.

This paper will describe in the next section other state-of-the-art systems that participated in the QALD competition. Section 3 presents the general architecture of our approach and describes each of its steps. The obtained results in the QALD-5 challenge are included in section 4, along with analysis for some of the questions the system answered incorrectly. In the end, we present the conclusions of this paper and some future development directions.

2 Related Work

As QAnswer uses a knowledge base for question answering (QA), the focus of this section will be on QA systems and approaches that employ ontologies or other sources of structured knowledge to generate the answer. One of the most important competitions for this type of systems is Question Answering over Linked Data - QALD[3], which is part of the Question Answering lab at CLEF. This competition, already at the fifth edition, provides valuable datasets with questions and answers that can be used to evaluate and compare such systems. For one of the tasks, the questions are chosen so that they could be answered by using DBpedia resources, so this task was perfectly suited to evaluate our system.

The most successful system in QALD-4, but also QALD-5, was Xser [6]. The system uses a two-layered architecture, where the first step produces a Directed Acyclic Graph (DAG) from the question with phrases labelled as resource, relation, type or variable, by using a structured perceptron [2]. These tags are independent of any knowledge base, so their approach can be very easily applied on a different ontology. The second step of their system maps the discovered entities to the ones in a given knowledge base, such as DBpedia, by using a Naïve Bayes approach. The downside of their approach is the need of a large annotated corpus of questions with DAGs.

Another interesting solution was used by gAnswer [14], which uses a graph-driven approach. The disambiguation takes place at the evaluation step, when the generated graph is matched with subgraphs in the ontology. While usually expressions are extracted for a given property, they use a database with common expressions [12] and then they map them to paths in the ontology. This way, more complex ontology constructs can be linked with expressions.

Casia [5] uses a similar approach for matching properties based on an expressions database. However, the system chooses between all possible phrases and their interpretation in the same step by using a Markov Logical Network [13].

There are also other types of systems for question answering, which do not use a knowledge base, but rather try to select the most appropriate answer for a given question from texts available online. This kind of systems are usually able to answer a greater variety of questions, but they are not as precise. The systems which combine these two strategies are called hybrid systems, and are probably the most successful. The best example is IBM Watson [3], who was able to beat the best human players in the Jeopardy game.

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

3 Proposed Solution

The system has a pipeline architecture, described in Fig. 1. The pipeline starts from the natural language question and ends with the answer.

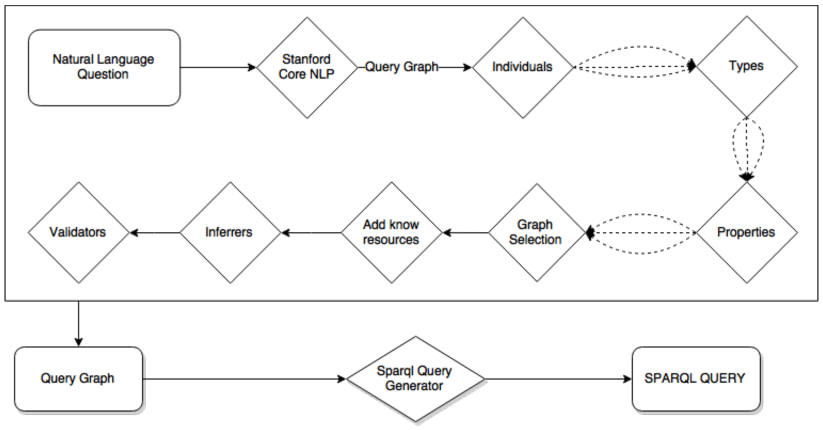

Fig. 1: The System Architecture

The answer is provided based on the generated SPARQL query, which is executed on a Virtuoso-opensource[4] endpoint, loaded with the DBpedia 2014 dump[5]. In order to be able to generate the query, two conditions are needed: the DBpedia entities must be discovered in text, and they have to be correctly linked. The links between the entities can only be based on syntactic dependency links between words. Because of this, the first step in the pipeline is the question parsing which is done using the Stanford CoreNLP library[6].

After this first step, a directed graph is generated, with vertices corresponding to tokens in the question, which are annotated with lemma and part-of-speech tags, and edges corresponding to the collapsed dependencies generated by Stanford CoreNLP [9]. Also, numbers and dates are detected in this step by using Stanford NER[7], a named entity recognition module.

Once the graph is generated, DBpedia entities must be detected. There are three types of entities in the ontology: individuals, types and properties. Methods for detecting each type of entity were developed, because of the particularities of each type. During these steps, multiple graphs will be generated because of the different possible matches, some of them containing multiple words. In the end, only one graph will be selected based on some scores as will be explained in the following sections.

[4] https://github.com/openlink/virtuoso-opensource
[5] http://downloads.dbpedia.org/2014/
[6] http://nlp.stanford.edu/software/corenlp.shtml
[7] http://nlp.stanford.edu/software/CRF-NER.shtml

The most probable graph enters the last stages of the pipeline, where missing entities are inferred, while existing ones are validated using the ontology. The SPARQL query is generated in the last step, creating triples and subqueries based on the structure of the graph and the direction of the properties.

3.1 Individual Detection

The detection of individuals from the ontology is probably the simplest task out of the three types of detection tasks, because the list of possible ways of expressing an individual is very limited. In the same time, it is a very important step, because individuals are much more specific than types and properties, which could be inferred in some cases based on the individuals discovered in this first stage.

In the general form, the named entity recognition (NER) task is a more difficult problem than the task at hand. This happens because the list of possible entities is usually open (almost anything can be a name). In our case, the entities must be part of DBpedia, any other match being useless. For each possible individual to match, most of the ways in which it can be expressed already exist in DBpedia, in the form of Wikipedia redirects. Although this approach might loose some expressions for certain individuals, it works well enough in most cases.

First, the individuals are filtered by their "importance", so we can ignore very unlikely candidates which can have a negative effect on the precision. The importance is estimated by the number of Wikipedia pages that have links to the individual's page. For each such individual, all the redirects that contain only ASCII characters are kept.

All these expressions are inserted in a trie data structure for words, in order to perform very fast searches. When searching for matches, the longest sequence of words that match is kept and for each sequence, the individuals are sorted based on how close their label is to the sequence of words, measured with the edit distance [7], and also by their importance.

Matched sequences of words can also overlap, each sequence being linked to a different individual. In this case, there is no way we can decide at this point which is the correct one, so the current graph is replaced by two different ones. Also, the same sequence of words might produce different graphs if the possible individuals that match have very different types. The type of each individual is important in the properties matching, which will be described later.

Another rule added to improve accuracy is the idea of a "strong" link. These links correspond to a sequence of more than one word, where each word that is not a preposition or conjunction is written with capital letter. These cases strongly indicate a correct match, so any other possible sequence that overlaps with it, or possible type or property match, will be ignored.

3.2 Type Detection

Although the number of types in DBpedia is much smaller than the number of individuals, their detection is harder because the ways of expressing them is not known a-priori, and because most of them are common words. Since for types we don't have Wikipedia redirects anymore, a new method of determining possible expressions needed to be found.

The "naïve" solution. Most of the times, a type appears in text by its label, which makes it very easy to discover. Starting from this, a database can be generated with entries containing pairs of types and word. Besides the actual label, synonyms of the label extracted from WordNet [10] are also added. This approach discovers many of the correct matches, but it is hard to apply on types whose label contain more than one word.

A Wikipedia-based approach. In order to overcome the shortcomings of the "naïve" approach, a more general solution was needed. Because of the strong connection between DBpedia and Wikipedia, each Wikipedia article corresponds to a DBpedia individual, who also has a type, which means that the page should contain a formulation of that type within the text. Luckily, the type is most of the times expressed in the first sentence of each article, which provides a short description for each individual.

Since the majority of the first sentences look like "<individual> is a <type>", the complement of the copulative verb should be an expression for that type. Some of the words connected with the complement of the copulative verb are also important for the expression of the type. Because of this, a subgraph containing vertices that can be reached starting from the type vertex by going only on edges corresponding to dependencies of a modifier type is generated. The words from this subgraph are memorized as a way of expressing the type.

We have found that, while some of the extra words are important for that type (e.g. "formula one racing driver"), others don't add any extra information (e.g. "English actor"). However, using all these possibilities might alter the results or might include in the type words that are important for the question in other ways. These problems and others are solved in a later step using a method that removes information that is too specific or too general.

The remaining entries are added in another trie-like structure, where edges can be lemmas or whole words. When queried, the word edges have priority, but lemmas are used when matches are not found. This permits, for example, the existence of the plural form for some words. Type detection is applied for each resulted graph from the individual detection phase, so at least the same number of graphs result from this step.

3.3 Property Detection

Out of the three types of DBpedia resources, properties are arguably the most difficult to detect. This is because there are more properties than types and most of them are complex and contain more than one word. Moreover, applying the same approach as for types is tedious because properties can appear anywhere in a Wikipedia article, and they can appear in many types of sentences, which makes developing rules for words extraction more difficult.

Extraction. Because of the strong connection to DBpedia, Wikipedia is a good source for pattern extraction, and was used for this purpose by systems like WikiFramework [8] and BOA [4]. We propose a similar approach, that also uses syntactic dependencies to select the pattern from the sentence.

Because dependency parsing is quite slow, only the sentences from a Wikipedia page where a property could occur were considered. This was achieved by finding sentences where both the label of the individual and the value of the property were found. For data properties, more possible values were looked for, especially for dates which have many possible formats. Although a lot of relevant sentences are ignored because the individual appears most of the times as a reference, not by label, using a coreference system would slow this task even more, and we concluded not to use coreference resolution at this point in our system.

From each selected sentence, the path between the discovered subject and object should be an expression that corresponds to the property between them. This is not always the case because other related words might be relevant for the meaning of the property, while sometimes, some words from the path might not be relevant. Starting from the path, other words, that could be reached from the ones on the path by going on edges that usually represent modifier dependencies, are added to the expression.

Sometimes, the extracted expression contains a lot of unnecessary words. An example of such a case is "*Judd Trump* is an English professional snooker player from *Bristol* and former world number one". This is a very common situation, where the individual is first defined, and then property value is connected to the definition. In this case, the individual is considered the word "player" so the extracted expression will be "from", which is much more relevant than "is an English professional snooker player from".

Matching. A subgraph is considered to match an expression if it has all the words in the expression, regardless of their order. Also, the words are compared by their lemma and POS tag, so all variations of a word can match. In this form, this problem is a subgraph matching one.

Finding all the possible matches is done by starting from each word in the sentence. An inverse index contains all the extracted expressions that contain a certain word, so a possible list of expressions is extracted fast. For each expression in the list, a search is performed starting from the vertex to check if all words in the expressions form a subgraph in the query graph. The validated expressions enter the next step of the algorithm.

A score is computed for each matched expression, based on the types that it connects. In the general case, the subgraph connects two words, which might have corresponding types. The type can be a detected type, the type of a detected individual or the type of a question word (who - Agent, when - Date, how many - Integer, etc.). Both types are then compared with the ones from the database in order to compute the score.

Considering that the types of the matched resource are (t_1, t_2) and the types from the database are $(t_1^{'}, t_2^{'})$, the distance between them is computed as the sum of the distances between $t_1$and $t_1^{'}$ and between t_2 and $t_2^{'}$. The distance between two types is computed as the number of nodes between them in the DBpedia taxonomy. There are two distinct taxonomies, one for resource classes, and one for data types. The distance between 2 classes from different taxonomies is considered maximum. Based on the computed distance between the pairs of types, the score of the matching is calculated with the formula (1):

$$S(e, tp, p) = \sum_{tp'} \frac{N(e, tp^{'}, p)}{N(e, tp')} * 2^{-dist(tp,tp^{'})} \tag{1}$$

where:

- e = the matched expression
- tp = the type pair (t_1, t_2)
- p = the property
- $tp^{'}$ = a type pair for property p and expression e in the database
- N = the number of appearances in the database for the given parameters

Because the same property appears in the database for different type pairs, the score of the match is summed for all occurrences. Also, any entry with the distance larger than a threshold is ignored.

In some cases, the expression in the sentence does not occur between two types. One such example is the question "How often did Jane Fonda marry?". Here, *marry* has only one connected type, which is *Person*, because *how many* cannot be linked with a type. Another similar case is represented by questions containing superlative adjectives, like "Which is the *highest* mountain?", where the words that are supposed to be matched to a property have only one neighbour in the graph. For these cases, results that match only one type are included in the list of matches, but with a lower score than matches with two types.

Since the matches contain only sets of vertices from the query graph, prepositions cannot be captured by this method because they are mapped as dependencies in the graph. It would be useful to capture such matches too, because they appear quite often in questions. In order to match edges too, the same scoring for pairs of types is used, for every edge in the graph that is not included in another match and that exists in the database.

Graphs Generation. In the case of individuals and types, the matching process selects the longest sequence at discovery time and all smaller included sequences are ignored. However, in the case of properties, the match consists of a subgraph, so it is more difficult to determine which matches to ignore. All the detected matches must be distributed to as few graphs as possible, while having as many non-overlapping matches in each graph.

Determining if two matches can be in the same graph can be easily computed by checking if their lists of replaced elements are disjoint. By comparing all pairs of matches, an undirected graph can be constructed with vertices being the matches, and the existence of an edge between two vertices meaning that the two matches do not overlap. All maximum cliques in this new graph represent the sets of matches that should be grouped in the same graph. However, the problem of finding all maximal cliques is NP-complete. Several algorithms for this problem exist, but probably the Bron-Kerbosch algorithm [1] is the most famous one. The Bron-Kerbosch algorithm has a worst case complexity of $O(3^{\frac{n}{3}})$, which matches the maximum number of maximal cliques in a graph, which is $3^{\frac{n}{3}}$ [11].

The number of properties in a question depends on the number of vertices in the input graph, but generally it is very unlikely to be more than a couple (e.g. maximum 3-4 properties). This means that the size of the graph on which the search for the maximal cliques is performed is very small in most situations, so applying the algorithm won't cause any performance issues, although there isn't a theoretic upper limit for the size of the graph. In the vast majority of cases, only one property is detected per input question, so this algorithm is not even run.

3.4 Graph Selection

The three separate detection steps produce more than one graph for complex questions, but testing every one of them on the ontology would take too much time. The rest of the steps in the pipeline are more complex because detected entities must be validated, while missing ones have to be inferred. All these actions require complex queries and the number of investigated graphs can be quite high. Also, the final results of each query are difficult to compare so it is hard to choose the best interpretation.

For all these reasons, QAnswer selects the highest scoring graph right after the matching steps and all the following steps will use only this graph. For each generated graph, a score has to be computed that reflects how well each match fits the text and how well the matches fit together. The first part is the sum of the scores for each matched entity. One problem is the fact that each type of property uses a different score formula because of the different matching methods applied. All these scores coming from different formulas are not trivially comparable because of the different range of the values, but also because some types of entities might be more important than others, depending on the question.

In the case of individuals, the only measure we can use is the importance of the entity in the DBpedia graph, which is computed as described in the previous sections. A logarithm is applied in order to make this score comparable to those for the other types. Also, the number of matched words increase the score, as it can be seen in the formula 2:

$$S_{ind}(I, match) = \ln(1 + importance(I)) * nWords(match) \tag{2}$$

The same formula is applied for types, only by replacing the importance of an individual with the number of occurrences of a type in our database:

$$S_{type}(T, match) = \ln(1 + occurences(T, match)) * nWords(match) \quad (3)$$

The property score is calculated as defined in formula 1. While the values for each pair of types in a property represents a probability, the sum over all existing pairs can produce results bigger than 1 for good matches, but usually lower than the scores for individuals or types. This is important because the certainty of a property match is lower than the one for the other types. Thus, we prefer to assign a lower importance to properties.

The graph score also contains bonuses for complete triples of (individual, property, individual) or (individual, property, type) that appear in the graph, because this account for a higher success rate. The final formula for the score is 4, where the triples are only the connected ones described above.

$$S(g) = \sum_{m \epsilon matches} S(m) + \sum_{(x,y,z) \epsilon triples} S(x) * S(y) * S(z) \quad (4)$$

3.5 SPARQL generation

The SPARQL generation is the last step in the pipeline. This component will receive a query graph with all the possible entities set on the nodes and it has the job to generate a valid SPARQL query based on this graph.

The process of generating the SPARQL query is done recursively from the root of the graph (the node with no incoming edges) downwards.

When a node is visited, it has access to the graph, the parent, the current SPARQL query and the triple constructed at the parent level.

At each step we take certain decisions of how to enrich the query based on the current triple that is being constructed, the current edge that has been visited and type of the current node.

Determining the answer type. In a QA system, the questions asked by the users can be of multiple types. They can ask for a list of things, what is the number of certain things, what is the date of an event, location questions etc.

The type of the question is determined by checking for patterns in the input. By default, we will list all the results of the SPARQL query. The other types of questions that we are checking for are the ones that need to count the number of results (e.g. :"How many . . . ?") or the ones that ask us to validate the query (e.g. "Is Berlin in Germany?").

An important part of the query generation process is accurately determining what the searched variable is. We are doing this process by analyzing the edges of the graph. Usually the searched element will be determined from the "nsubj" edges in the dependency graph. From the connected resources of this edge we look for edges that have a "det" (determiner) dependency connected to them. This highlights that the source connected to them is the one being searched.

4 Results

The system was evaluated on the QALD-5 test corpus. Unlike other systems, ours did not use any training on the data so any differences between the results on the training and test data are probably due to the difficulty of the questions, and not the result of overfitting. However, any testing and changes of the system's parameters were performed on the training data and this may have caused overfitting on this training set. On the other hand, the results on the training set were slightly worse than those on the test set with about 0.27 global F1 score on test set compared to 0.30 global F1 on test set.

Table 1 presents the results of all the participating systems in the multilingual question answering over DBpedia task. Taking into account the global F-1 score, our system obtained the second best result after Xser, who outperformed all the other systems by far.

Table 1: QALD-5 results

	Processed	Right	Partial	Recall	Precision	F1	F1 Global
Xser (en)	42	26	7	0.72	0.74	0.73	0.63
QAnswer (en)	37	9	4	0.35	0.46	0.40	0.30
APEQ (en)	26	8	5	0.48	0.40	0.44	0.23
SemGraphQA (en)	31	7	3	0.32	0.31	0.31	0.20
YodaQA (en)	33	8	2	0.25	0.28	0.26	0.18

Out of the 50 test questions, QAnswer only tried to answer those questions marked as "onlydbo" because possible expressions for types and properties were extracted only for the "dbpedia.org/ontology" ones. The system returns an answer for each of those questions, because it is hard to determine when an interpretation is incorrect.

4.1 Questions Interpreted Incorrectly

The system might fail to produce the correct answer due to several reasons. Below, we present some of the problems identified after interpreting the results on the questions from the QALD-5 test dataset.

- "Which programming languages were influenced by Perl?"

Our system detected all the needed entities in the question, but generated a query with the wrong direction of the property *influenced*. This is expected to happen, since the property matching algorithm does not take into account the direction, which could be very clear in this case due to the word *by*.

- "Who killed John Lennon?"

The question was labeled with *onlydbo*, but the information needed to answer the question required the *dbpedia.org/property/conviction* property. Also, this question is very difficult to answer because the entity that needed to be detected was *Death_ of_ John_ Lennon*, and not the much more probable *John_ Lennon.*

– "Which artists were born on the same date as Rachel Stevens?"

Although all the required entities were matched, our system is not yet able to answer questions where a comparison on dates is needed.

– "Who is the manager of Real Madrid?"

Because of the long extraction process for properties, we have only searched for those properties that appeared in previous QALD datasets. The idea that those were the most important properties was not really true, as many new properties appeared in the training and test set. Those new properties cannot be detected by our system, so questions containing them, like the one above, were not interpreted correctly. We shall solve this problem in the next version of the system.

– "Give me the currency of China."

Sometimes, the graph scoring formula indicated a wrong interpretation as the best one. In this case, *currency* was labeled as a type, instead of the property. In order to solve problems like this one, other ways of evaluating an interpretation must be found, or more than one graph should be evaluated.

5 Conclusions

We developed QAnswer - a question answering system that uses expressions extracted from Wikipedia to match entities in DBpedia. The solution can be generalized to other ontologies, but the properties extraction step must be executed again for the properties in these new ontologies. More, if an individual is not part of DBpedia, new text sources that describe it should be identified, since this means that it does not have a Wikipedia page either.

At the time the system was tested on the QALD-5 test dataset, only expressions for a small part of the properties in DBpedia were extracted. The process should be executed for other common properties too, so that a more relevant evaluation of our method can be performed. Also, the extraction process can be improved because a lot of irrelevant information appear in the extracted expressions, which may affect the results.

Methods for learning which graph interpretation is more plausible will be developed and tested in the future. Such methods could help us detect the correct type of an entity for a phrase, and not depend too much on graph scoring, which cannot reflect whether an interpretation makes sense or not.

Acknowledgements. This work has been partly funded by the Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of European Funds through the Financial Agreements POSDRU/159/1.5/S/132397 and by POSDRU/155420 - PROSCIENCE. Stefan Ruseti has also been awarded a student travel grant for CLEF 2015.

References

1. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM 16(9), 575–577 (Sep 1973), http://dl.acm.org/citation.cfm?id=362342.362367
2. Collins, M.: Discriminative training methods for hidden Markov models. In: Proceedings of the ACL-02 conference on Empirical methods in natural language processing - EMNLP '02. vol. 10, pp. 1–8. Association for Computational Linguistics, Morristown, NJ, USA (Jul 2002), http://dl.acm.org/citation.cfm?id=1118693.1118694
3. Ferrucci, D., Brown, E., Chu-Carroll, J., Fan, J.: Building Watson: An overview of the DeepQA project. AI magazine pp. 59–79 (2010), http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/2303
4. Gerber, D., Ngonga Ngomo, A.C.: Bootstrapping the Linked Data Web. 1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011 1 (2011)
5. He, S., Zhang, Y., Liu, K., Zhao, J.: CASIA @ V2 : A MLN-based Question Answering System over Linked Data. In: CLEF 2014 Working Notes Papers. pp. 1249–1259. No. 61272332 (2014)
6. Kun Xu, Sheng Zhang, Yansong Feng, Zhao, D.: Answering Natural Language Questions via Phrasal Semantic Parsing. In: CLEF 2014 Working Notes Papers. pp. 333–344 (2014)
7. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady 10, 707 (Feb 1966), http://adsabs.harvard.edu/abs/1966SPhD...10..707L
8. Mahendra, R., Wanzare, L., Bernardi, R.: Acquiring Relational Patterns from Wikipedia: A Case Study. Proceedings of the 5th Language and Technology Conference pp. 111–115 (2011), http://disi.unitn.it/ bernardi/Papers/ltc.pdf
9. Marneffe, M.D., Manning, C.: Stanford typed dependencies manual (September 2008), 1–28 (2008), http://nlp.stanford.edu/downloads/dependencies_manual.pdf
10. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.J.: Introduction to WordNet: An On-line Lexical Database *. International Journal of Lexicography 3(4), 235–244 (Jan 1990), http://ijl.oxfordjournals.org/content/3/4/235.short
11. Moon, J.W., Moser, L.: On cliques in graphs. Israel Journal of Mathematics 3(1), 23–28 (Mar 1965), http://link.springer.com/10.1007/BF02760024
12. Nakashole, N., Weikum, G., Suchanek, F.M.: PATTY: A Taxonomy of Relational Patterns with Semantic Types. EMNLP-CoNLL pp. 1135–1145 (2012), http://dblp.uni-trier.de/db/conf/emnlp/emnlp2012.html#NakasholeWS12
13. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (Jan 2006), http://link.springer.com/10.1007/s10994-006-5833-1
14. Zou, L., Huang, R., Wang, H., Yu, J.X., He, W., Zhao, D.: Natural language question answering over RDF: a graph data driven approach. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14. pp. 313–324. ACM Press, New York, New York, USA (Jun 2014), http://dl.acm.org/citation.cfm?id=2588555.2610525