Keyword and Constraint Identification for Question Answering

Prathyusha Jwalapuram and Radhika Mamidi

Language Technologies Research Center
International Institute of Information Technology
Hyderabad, India

Abstract. The paper discusses a simple, rule-based system of extracting relevant information from a user's query. The idea is to extract this information by using only dependency relations and POS tags, obtained from the Stanford Dependency Parser. Using the universal dependency tags provided by the parser, we try to understand the semantic structure of a query. This is done by looking for semantically important dependents of the verb, such as the subject, direct object, prepositions and their objects, and so on. Using information obtained from a combination of the dependency relations and the inherent semantic implications of words (such as 'who' or 'where'), we try to extract the main objective of the query, and the constraints pertaining to it. The implementation itself is domain independent, however, a mapping to a knowledge base would require some domain knowledge. Some examples of the implementation over a question corpus for a library and a corpus of questions designed for a course management portal in the academic domain, created previously by [1] is discussed.

1 Introduction

Table 1. Example of Ambiguity

Q: Who takes the NLP course?
<< find: professor; constraint: NLP >> take = teach
Q: Who has taken the NLP course?
<< find: students; constraint: NLP >> take = register

In order to provide an effective question answering system, it is important to be accessible to the common user. The common user does not know a query language or programming or understand templates and formats. In order to be accessible to this common user, the system must be able to understand a natural language query. For example, for a user to be able to access data from a database, not only does the user need to know a query language used by the database, but the user needs to have a thorough knowledge of the database schema as well in order to be able to formulate a query based on the information they seek. Most users don't have the requisite knowledge.

The idea behind the interface is to allow users to ask questions in everyday, conversational language, and process these questions to convert them to a valid query in a language understood by the database (such as SQL) or any other knowledge representation system (such as an ontology) in order to obtain results, and thereby answer the user's questions. The typical information retrieval approach is to filter out function words and use the noun phrases as keywords; here we show that significant information can be gained by inferring information from function words.

However, since conversational language is fraught with ambiguities, and users may ask the same question in many different ways, the task of breaking down a user's query to understand what they are looking for is not easy. Often, such ambiguities cannot be resolved without contextual information, which is not available in a single query question-answering system.

In the example in Table 1, the questions *'Who takes the NLP course? / Who has taken the NLP course?'* could both be asking about the professor who teaches the course or the students who usually register for it. Although this ambiguity cannot be resolved without additional context, we hope to at least extract the information that the user in this case is looking for a person or persons (indicated by *who*), who is the agent of *take* where the course name is *NLP*. The keywords *who, take* and *NLP* are extracted with the help of universal dependency tags produced by the Stanford Parser; the association of *take* with either *register* or *teach* and the identification of *NLP* as a *course name* would require domain specific and in fact knowledge base specific information.

2 Related Work

One of the easiest approaches is to run a keyword or pattern matching algorithm that looks for domain specific keywords and patterns, and matches them to a predefined set of queries [2]. The limitation of this approach is that a large number of keywords and patterns must be anticipated, due to the large number of possible variations in language. This approach also tends to perform poorly in case of complex queries.

Systems such as PLANES [3], PRECISE [4] and STEP [5] opt for frames or patterns based on semantic grammar or, as in the case of STEP, a phrasal lexicon developed over the database. As in the case of keywords, this approach is not only domain dependent, but also relies heavily on a pre-constructed list, which is restrictive and therefore needs to be extensive in order to be practically useful. Such systems also tend to be accurate for semantically tractable queries.

A rule-based system was proposed by [6] that was based on the framework of Computational Paninian Grammar [7] to identify the semantic templates that a query may belong to. This approach uses domain specific semantic mapping, wherein Paninian grammar constructs are mapped to dependency relations. However, it requires the creation of verb frames with specified arguments, making the query structure rigid. All possible argument structures must be identified for all possible verbs in the domain, and each of these arguments must be mapped to the relevant entities they refer to in the knowledge base. Combined with the introduction of parser errors, this approach can work well with limited types of questions.

Another approach is to use machine learning algorithms to generate a statistical model, as explored in [1] where the focus is on identifying the main concepts required for query generation, using a specific tag-set and a CRF++ algorithm. The algorithm tries to identify the concepts and map them directly to the database attributes they refer to. As with any machine learning approach, the system is dependent on the quality of the training data, and any changes that are made require re-training with new data. However, this system shows a high precision for a domain specific system, in this case a course management portal in the academic domain.

3 Our Approach

Our emphasis was to try and identify the objective of a user's query, that is, the data the user is looking for and the constraints pertaining to this data. The idea was to do this using as few tools as possible. We only use the Stanford POS Tagger and the Stanford Dependency Parser for the implementation of our system (see Fig.1).

We assume that the queries are free of errors and are grammatically sound, and that all required arguments are present in the query. We also don't attempt to resolve abbreviations and ambiguities.

The system is able to extract information from ungrammatical queries in certain cases where the Stanford Parser is able to generate a fairly accurate dependency parse. Since the aim was to identify the objective and constraints from the query, we did not test the mapping from the obtained information to a knowledge representation system (such as a database), but we explore the possibility.

3.1 Motivation behind using Dependency Relations

The advantage of using dependency relations is that they are syntacto-semantic relations. This makes them word-order independent. The same question formulated in different ways can lead to similar dependency relations (such as a question in active/passive voice); this allows us to easily group similar user queries without having to anticipate all the possibilities.

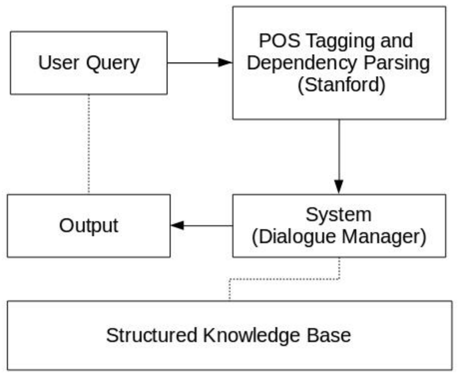
User Query
POS Tagging and Dependency Parsing (Stanford)
Output
System (Dialogue Manager)
Structured Knowledge Base

Fig. 1. System Architecture

Dependency relations also give us an idea of the relationship between the words and therefore the information the user is looking for and the constraints. This information cannot be obtained from simple syntactic parsing, such as Phrase Structure Trees, as they are highly dependent on the word order.

3.2 Dealing with Parsing Errors

After obtaining the dependency relations and parts of speech of a given query, we try and construct some simple understanding of the query. It was found that the Stanford Dependency Parser does not always tag the main verb of the sentence as ROOT, and tends to tag question words or sometimes other nouns in the query as ROOT, especially in queries containing a copula. This also leads to identification of the direct object and other relations incorrectly, thereby making any mapping using specific tags inaccurate.

However, the overall structure identified between the words of the sentence is usually correct. In order to avoid the errors introduced by faulty tags, we crosscheck the assigned tags with the part-of-speech of the word, and in case of any discrepancy, we disregard the tags assigned to the words. For example, in case of questions like *'What is X'*, the parser may tag *What* as the root. Since *What* would have a POS tag of WP, we disregard the ROOT tag, and instead use only the structure of the sentence in terms of word dependencies, along with POS information.

This approach for dealing with parsing errors is only useful for short questions; for longer questions with multiple verbs and nouns significant information is lost. This means that often dependency relations are missing and wrong tags are assigned to words, which introduces significant error.

3.3 Processing the Parser Output

Prepositions and their Arguments In order to find relevant information, the system looks for the children of the root that are nouns, adjectives pertaining to nouns, prepositions and their children, and so on.

Prepositions such as *'by', 'on', 'of'*, etc. provide useful information about the relations and constraints. For example, although *'on'* might also have a locational meaning attached to it, in the context of a library QA we can assume it shows reference to a topic (*'What books on image processing are available?'*); similarly for the prepositions *about* and *under.*

Prepositions such as *in* and *at* often convey spatial or temporal information. Their noun dependents, along with the prepositions themselves, are considered to be constraints.

Stanford parser identifies the preposition *by* and its dependent as agents; this information is also added to the constraint list. For example, consider *'What are the books written by Ayn Rand?'*. Here the user is looking for books where the *author* is *Ayn Rand.* The preposition *by* here indicates that *Ayn Rand* is the agent of *'write'*, which in the domain of library translates to *'author'*.

In fact, in almost all cases, the noun dependent of the preposition turns out to be a crucial constraint. However, there are a few cases in questions like *'What books by Jeffrey Archer are available in the library?'*, that have to be identified as irrelevant information since in the library is technically implied and provides us with no useful information – and filtered out. But this is a domain specific assumption, and there is no way to judge the relevance otherwise.

Question Words We look for the question words in the query if any, and use them to further identify the user's intent. For example, queries using *'who'* are looking for a person (in case of a library corpus, perhaps the author, or in case of the courses domain, perhaps a student or a professor), queries using *'where'* are looking for a location, and so on.

These interpretations are domain dependent, so we could simply associate *'who'* with a person and *'where'* with a location, and have them independently converted to student/author and shelf number and so on in the stage where the keywords are being mapped to a knowledge base.

In the case of *'which'*, it appears as a determiner in questions, and the noun it modifies can be directly considered to be the objective of the question (*'Which* **book** *is...' or 'In which* **year**...') .

We also note the presence of a construction *'how many'*, which is looking for a numerical answer, which overrides any other objective identified in the question (which is relegated to a constraint).

Questions with *'how'* or *'why'* are not considered, as they would typically elicit subjective answers. However, questions themselves are not classified. Separate rules are required for questions without question words, which use modals or auxiliary verbs for question formation.

Yes/No Questions We identify the questions which are formed using constructions of auxiliary and modal verbs, which are looking for a yes/no answer. Such questions are structured slightly differently and require a different set of rules. Instead of looking for an objective - the answer type is clearly known to us here - we consider the *nsubj* (nominal subject), the *dobj* (direct object) and any constraints identified, all together as constraints.

This could introduce some difficulty during mapping to a knowledge representation, but the simplicity lies in the fact that if there is an instance that satisfies all the listed constraints, we can return an answer of *yes*, else we can return a *no*.

However, there are illocutionary implicatures to think about; for example, consider *'Do you have research papers related to NLP?'* Here, the answer is technically yes or no, but the user really wants a *list* of research papers related to NLP, as opposed to a question like *'Is the Fountainhead available for issuing?'* Since this distinction is pragmatic, we do not attempt to resolve it.

Modifiers and Conjunctions In order to find constraints, first we look at all the modifiers of the *nsubj/dobj* that has been identified. Any adjectives associated with the *nubj/dobj* are considered to be constraints.

Typically, the *amod* (adjectival modifier) tags are related to the words that directly modify the word and should be part of the left side of the query (such as **latest** *edition*) and the words tagged *nmod* (noun modifier) are part of the right side of the query equation (such as **Digit** *magazine*). The idea of a query equation if explained in further detail in section 4.

We group the nouns related through the compound tags (such as *Natural Language Processing, Data Structures,* etc). This kind of grouping may fail in certain cases where conjunctions are involved for example, *SSAD and Project* is one single course name, but is treated as two different words by the parser, such that any preposition preceding it has both of them separately as children. For better grouping of such titles, a chunker may be used.

Temporal Information Certain temporal pragmatic information for example, deictic constructions such as *last week*, *till date*, etc, must be identified and processed.

Consider *'Which is the latest edition of Digit magazine available?'*. Here, latest would require to be interpreted as

<<find edition magazine=Digit, date=max(date)>>

Here we see that though *latest* is a modifier for edition, it really needs to be about the date; if the modifier had been *oldest* or *first*, we should be looking for *date = min(date)* instead. Specific constructions (such as *'in the year 2000'*) are taken care of as part of the information extracted from prepositions and their dependents.

Verbs and their modifiers We do not consider verbs and their modifiers since we place very little emphasis on the information provided by verbs, except in the case where agency of an act is a constraint, like in the case of *'Who wrote The Fountainhead?'* where we are looking for a person who is the agent of the action *'write'*; here the verb provides significant information that tells us that the user is looking for the *author*.

In case of questions that do not use question words, and instead use constructions such as *'List X'* or *'Show me X'*, the requisite information obtained by the system will be the same, and these verbs do not add much value to the information.

In the cases where the verb is considered important, significant information provided by adverbial modifiers specially negations must be specially considered. The problem here is that the negation itself would be associated with the verb, but the conversion of the query into the keyword and constraints format requires the negation to move to a different keyword. Consider, *'List all the books which are not written by Ayn Rand'*. Here, the action is *'not written'*, and the *neg* tag is associated with written; however, the keyword and constraints must look like: *find books, author = not(Ayn Rand).*

Example Consider the query, *'List the students registered for honours projects under Professor XYZ'*.

A simplified dependency tree constructed from the tags provided by the Stanford Dependency Parser would look like Fig.2. In this case, the information the user is looking for (*'students'*) is the direct object, and the constraints are directly related to it.

In order to identify this main objective of the question, we identify the direct object of the main verb (root) of the question, tagged as *dobj* by the Stanford parser. In certain cases, such as passive constructions, this is often tagged as *nsubjpass* (subject, passive) (henceforth we refer to both as *dobj*). In cases without a marked *dobj*, we look for other noun dependents of the verb such as *nsubj* for constraints. In the case of questions with *'who'*, the requisite information is an agent, and the *dobj* is merely a constraint. This information is fairly obvious and can be inferred directly from the question word.

Domain Independence This architecture of the system is domain independent, and any dependencies must arise only from a required mapping of the keyword and constraints to a knowledge system. For example, mapping a *'who'* to a *student* is specific to a *courses* domain, while mapping a *'who'* to an *author* would be specific to a *library* domain. For this reason, we leave semantically significant question words such as *'who'* and *'where'* undifferentiated, and merely represented as *person* and *location* respectively.

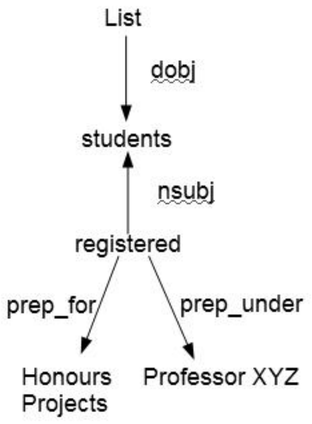

Fig. 2. Dependency Tree of a Sample Query

Table 2. Query Equations

Q: List the books written by Ayn Rand.
<< find books, written_by = Ayn Rand >> Here *'written_by'* must be mapped to *'author'*.
Q: Which is the latest edition of Digit magazine available?
<< find edition, magazine=digit, date=max(date) >> Here, *latest* is an *amod*, and is resolved to mean *max(date)*; the modifier *Digit* is an *nmod* and is therefore equated as *magazine=Digit.*
Q: Do you have research papers related to NLP?
<< find research papers, topic = NLP >> Here *related to* is being interpreted as *topic*; in fact the preposition *'to'* by itself is used to arrive at this, so that the dependence on having to consider all possibilities (*related to, relevant to,* etc.) is reduced.
Q: What books on socialism are available in the library?
<< find books, topic = socialism >> Here *'in the library'* is ignored, as it should be, but this is a domain specific choice that is particularly filtered out. In any other case, it may appear as an additional constraint (as *location=library*, for example)

4 Query Equation

Once all the primary keywords and the relevant constraints have been identified, we build a pseudo-SQL query like output to properly group the constraints. The final output is merely for representational ease; the output itself is a collection of keywords and related constraints, which can be mapped to any kind of knowledge representation. However, since an SQL-like representation would be the easiest to understand, we chose to demonstrate the output in that format. Since it is not really SQL, and the terms are not mapped to any relations, we choose to call it a query equation (mainly because we mostly equate some value as a constraint)
.

Examples of query equations are given in Table 2.

As can be seen, the query equation is in the form of a pseudo-SQL query; it can be easily converted into a format that suits any knowledge representation, provided the relations that the terms map to are known. Consider the first example,

List the books written by Ayn Rand.
<<find books, written_by = Ayn Rand >>

This can be mapped to an SQL query, say a relation *books* with attributes *book number, title, author, publisher, year,* etc. as follows:

SELECT title FROM books WHERE author like 'Ayn Rand';

5 Evaluation

5.1 Accuracy

Table 3. Results

Corpus	**Accuracy**
Library Domain	72.72%
Library Domain without questions with relative clauses	85.7%
Course-management Domain	61.1%

The system was tested on a set of 128 questions for library domain obtained through a survey, and another set of 150 questions on the courses domain created by [1].

Since the emphasis is on identifying the right constraints, the evaluation is based purely on whether the intended keywords and any associated constraints are correctly identified from the question. A question's objective is considered to be rightly identified only if both what the user is looking for and all the necessary information related to it is identified correctly.

As is summarized in Table 3, the accuracy of the system increases when the questions which have relative clause descriptions and therefore multiple verbs (e.g. *'What other books does the library have that were written by the author of the Da Vinci Code?'*) are not considered.

There is a significant drop in the accuracy when tested on a course management domain; the level of ambiguity in these questions was found to be higher, along with far fewer preposition dependent constraints and more relative clause questions.

5.2 Error Analysis

Apart from parser errors, there are many other errors introduced due to oversight, ambiguous structures, grammatically unsound sentences, etc.

Conjunction - 'and' One of the significant problems occurs with conjunctions such as *'and'* that link two relevant words together. Consider *'Harry Potter and the Philosopher's Stone'*; in this case, *Harry Potter* and *Philosopher's Stone* are parsed as two separate arguments linked by a conjunction, but they are in fact part of the same title. In such cases, it is difficult to tell if the user has two different constraints or just one (*'I want books by Ayn Rand and Dan Brown'*) if we consider them separately, a search for *Harry Potter* would list all the seven books of the series as relevant, instead of just the first book. In certain cases, only one of them might be relevant; in such cases there has been no way to resolve the issue. Also consider cases with multiple unrelated constraints separated by conjunctions: for example, *'Which students are the TAs of NLP?'* and 'Which students are TAs and are registered for NLP?'.

Syntactic Parsing Issues Constructions such as dangling prepositions (*'How many credits is NLP for?', 'Which Y does X belong to?'*) lead to parser errors or unexpected dependencies that are not accounted for. Sometimes certain words are skipped by the parser.

More significantly, there is a PP attachment problem - consider the following queries:
(a) Who is the author of The Wind in the Willows?
(b) Is The Fountainhead present in the library?
(c) Who is the author of Body in the Library?

In (a), the PP *in the Willows* is part of the title, while in (b), *in the library* is irrelevant information, which cannot really be filtered domain specifically either, as demonstrated in (c) where *in the Library* is part of the title. Therefore, gauging the relevance of an argument is a challenge; there are too many possibilities that cannot all be taken into account individually.

This issue is not limited to prepositions alone; consider *'What is the ISBN of the book Talent is Overrated?'*, where the *is* belonging to the book title is recognized as a separate verb copula and is assigned arguments.

Pragmatics Resolving pragmatic issues is also a challenge. Consider *'Which is the most popular book on mathematics?'*. Here, we can extract information on the modifiers as
<<find most(popular(book) , topic = mathematics >>

However, converting *'the most popular'* to mean that we need to group by book title, then count the number of times each book has been *issued* and then return the one with the highest count, is quite a challenge; we need to infer that *popular* here is a reference to the *number of times a book has been issued*, which requires deriving information that is not present in the query.

Relative Clauses Any arguments that are expressed through non-prepositional phrases, such as through relative clauses, which bring in additional verb and its arguments, are less likely to be identified correctly as relevant constraints. The accuracy of the system improves when such queries are not considered.

Consider *'What other books does the library have that were written by the author of the Da Vinci Code?'*. There are two parts to the query; firstly we need to find the *author* of the *Da Vinci Code*, then find *books* written by that author, and also remove the book *The Da Vinci Code* itself and display the results. The query is complex and there are pragmatic issues as well.

6 Future Work

The system provides a simple framework requiring very little coding in order to extract relevant information from a user's query. Pre-processing problems involving spelling and grammatical errors, synonyms, negations, missing arguments, abbreviations, etc. need to be handled in order to make the system practically useful. Also, the verbs must be lemmatized and eventually replaced with a mapped, representative synonym that is part of the knowledge base taught, instruct, take, etc. must all map to teach, for example.

In order to find complete constraints, such as the full name of the Professor or the full course name (like *'Linear Algebra and its Applications'*), dependency relations are sometimes inadequate, since the tags often resolve to separate determiners and other function words which are often part of the title. This information can be captured using a chunker. The chunks provided often correspond to the entire title or name, providing us with the complete constraint.

It is important to handle verbs and verb modifiers in order to present a complete picture of the objective of the user. These must necessarily be handled. Handling multiple verbs and multiple main keywords is also necessary, as in the case of relative clauses. Our system currently only handles single line queries; multi-line queries are quite common and provide context spanning over two or more sentences which must be taken into account in order to extract relevant information (*'I like stories about dragons. Are there any such fantasy series? I've already read A Song of Ice and Fire'*).

Conjunctions are a very common occurrence, and multiple arguments must be considered. However, it may be difficult to identify a more relevant argument from a less relevant one.

To resolve intention recognition issues, such as in the case of *Do you have research papers related to NLP?*, which is technically a yes/no question but the user intention is really to get a list of research papers, a corpus that has been annotated with dialogue acts can be used.

A corpus with a wide variation in the pattern of questions, along with questions in statement form must be analyzed and considered, in order to make the system more accurate for generic queries.

The usefulness of the information gathered through this method and the ease of mapping to a data representation must also be studied.

The possibility of increasing the accuracy by using statistical machine learning methods also needs to be explored.

References

1. A. Palakurthi, R. Chandibhamar, S. Srirampur, R. Mamidi, "Concepts Identification of an NL query in NLIDB Systems", International Conference on Asian Language Processing (IALP), 2014.
2. T. Johnson, Natural language computing: the commercial applications, The Knowledge Engineering Review, vol. 1, no. 03, pp. 1123, 1984.
3. B. A. Goodman and D. L. Waltz , "Writing A Natural Languge Database System", Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1 I, Pages 144-150, 1977.
4. A. Popescu, O. Etzioni, and H. Kautz,Towards a theory of natural language interfaces to databases, in Proceedings of the 8th international conference on Intelligent user interfaces. ACM, pp.149157, 2003.
5. M. Minock, "A Phrasal Approach to Natural Language Interfaces over Databases", Proceedings of the International Conference on Applications of Natural Language to Information Systems (NLDB), pages 333-336, June 2005.
6. A. Gupta, A. R. Akula, D. K. Malladi, P. Kukkadapu, V. Ainavolu, R. Sangal, "A Novel Approach Towards Building a Portable NLIDB System Using the Computational Paninian Grammar Framework", International Conference on Asian Language Processing (IALP), 2012
7. A. Bharati, A. K. Gupta and R. Sangal, "Parsing Paninian Grammar with nesting constraints", Proceedings of 3rd NLP Pacific Rim Symposium, Seoul, S. Korea, 4-6 December 1995.