RACAI's Question Answering System at QA@CLEF 2007

Dan Tufiş, Dan Ştefănescu, Radu Ion, Alexandru Ceauşu

Institute for Artificial Intelligence, Romanian Academy
13, "13 Septembrie", 050711, Bucharest 5, Romania
{tufis, danstef, radu, aceausu}@racai.ro

Abstract. This paper presents a pattern-based question answering system for the Romanian-Romanian task of the Multiple Language Question Answering (QA@CLEF) track of the CLEF 2007 campaign. We show that working with a good Boolean searching engine and using question type driven answer extraction heuristics, one can achieve acceptable results (30% overall accuracy) using simple, pattern-based techniques. Furthermore, we will present an answer extraction algorithm that aims at finding the correct answer irrespective of the question and answer type.

Keywords: question answering, QA, text preprocessing, tokenization, POS tagging, document indexing, document retrieval, Lucene, answer extraction, query formulation

1 Introduction

Question Answering (QA) is a long-standing goal of Natural Language Processing (NLP) field of Artificial Intelligence. Currently it is viewed as an almost indispensable tool of Information Extraction (IE) with which one can seek the exact answers to the natural language questions that potentially need immediate attention using a large collection of documents. QA shifted the focus from Information Retrieval (IR), which gathers user's documents of interest in response to a specific query to IE and answer searching and extraction. This way, users are relieved of formulating complex queries to retrieve the documents in which they will have to look the answer for themselves. This last operation can be time consuming and because of it, most of the time, users of simple IR engines easily give up the search if they cannot find the answer in the first few passages of the top (at most 10) document hits of the engine.

In this context, QA engines offer the much expected way out for the users: formulate the queries in natural language and offer an answer or a list of answers from the retrieved list of candidate documents. To do this, the typical QA system implements the following components:

- a question analysis component which usually identifies the type of the question (factoid, definition, enumeration, etc.) and the type of the expected answer (specifically for factoid questions: person, location, date, organization, etc.);

- a paragraph extraction and ranking component which is responsible for selection of those paragraphs that may contain the sought answer to the user's query extracted from the documents that were returned by the text searching engine;
- an answer extraction module which scans the list of best ranked paragraphs in order to retrieve the complete, minimal and syntactically well-formed string(s) that constitute(s) the answer(s) to the user's natural language query.

All of the three components above are by no means easy to implement nor have they already been successfully implemented by the hard to meet standards of the human evaluation. To solve the QA problem, one has a multitude of natural language "problems" to deal with. Some of these such as part of speech tagging or syntactic analysis received satisfactory algorithmic solutions but others, ranging mainly in the semantic realm of natural language, may still receive better solutions to meet human level acceptance. Out of the latter problems, meaning equivalence and textual entailment between different natural language expressions is of outmost importance to QA. Consider for instance the question "*When did Elvis Presley die?*" and the text snippet "*The King of Rock & Roll died on August 16, 1977*". If only the QA algorithm could "know" that the "King of Rock & Roll" is a denomination of Elvis Presley, it could answer the question.

Current research in QA acknowledges these problems and it is no wonder that modern QA systems are almost incredibly complex comprising modules that deal with different levels of natural language representations. The semantic processing is playing a central role in answer determination. Systems such as FALCON ([3]), COGEX ([7]), PowerAnswer ([2,6]), LaSIE ([1]) and QUARK ([11]) use some form of logical representation of both question and candidate answers so as to be able to logically prove that the selected answers can be justified in terms of the question premises. The most recently published results on the QA track from the Text REtrieval Conference (TREC)[1], and also the results published in the previous years, show that QA systems using abduction as a form of validating answers are the best. For instance, PowerAnswer 3 achieved an accuracy of 0.578 for factoid questions at TREC 2006 ([6]) and PowerAnswer 2 scored 0.713 at TREC 2005 ([10]) for the same type of questions.

Although semantic processing is the next logical step in improving QA systems (and indeed any NLP system), occasionally QA may get away without it. For instance, LASSO ([8]) is the QA system that preceded FALCON and PowerAnswer and the answer was extracted using the parse tree of the candidate sentence and different heuristics to match keywords from the questions with those of the sentence. We call these types of QA systems, pattern or template-based QA systems.

In this paper we will describe a pattern-based QA system, which is a combination of two separate QA systems that use the same text search engine and different question analysis and answer extraction modules. Our system (denoted with the "ICIA" indicative in the runs we have submitted), participated in the Romanian QA task at the QA@CLEF 2007 where it obtained a 30% global accuracy. In what follows, we will give a description of each QA system along with their combination

[1] http://trec.nist.gov/pubs/trec15/t15_proceedings.html

and we will also describe our text searching engine that is a Boolean query engine based on the open source Apache Lucene[2] full text searching engine.

2 The Document Collection

The document collection was composed of 43486 Romanian language documents from Wikipedia[3]. The files provided for the competition were available both in HTML and XML formats (http://ilps.science.uva.nl/WikiXML/). We chose to use the XML files due to their *well-formed* and *valid* properties. As the original XML contained a lot of pictures, tables and other elements which were not relevant for the QA task, we transformed the files, using a XSLT schema, into XMLs preserving only the relevant information needed for IR and IE, organized and structured in such a way so as to be easily exploited both by Lucene and the IE tools we developed. The new files obtained have the following structure:

```
<wiki pgname="…" …>
    <title>…</title>
    <article>
        <title>…</title>
        <overview> <p>…</p> … <p>…</p> </overview>
        <section> <title>…</title>
            <content> <p>…</p> … <p>…</p> </content>
        </section>
    </article>
    <clinks>
        <cat pageid = "…"> … </cat>
        …
        <cat pageid = "…"> … </cat>
    </clinks>
</wiki>
```

Under the *clinks* tag we put the articles related to the current one. The titles and contents from each document in the collection were preprocessed so as to obtain sentence and word splitting, part of speech tagging (POS tagging) and lemmatisation using the Tokenizing, Tagging and Lemmatizing free running text module (TTL[4], [4]). TTL also incorporates a regular expression based named entity recognizer but due to its inceptive status, it has not been used. Thus, we rely on the output of the part of speech tagger for rough entity recognition: numeral (integer or real) and proper names, which are likely candidates for factoid questions.

After the TTL run, we parsed the entire document collection using our link analyser LexPar ([5]). The link analyser, or linker, does a dependency-like analysis of every sentence of every document in the collection. This dependency-like analysis is called a linkage and it is produced with a link filter adaptation (LexPar) of the Lexical Attraction Models (LAM) of Deniz Yuret ([12]). In principle, a LAM tries to assign the most likely undirected, acyclic, planar and connected graph to one sentence given that the vertexes of this graph are the words of the sentence and its edges are the dependency relations that hold between the matched words pairs. To exemplify the

[2] http://lucene.apache.org/
[3] http://ro.wikipedia.org/
[4] Also available online at http://nlp.racai.ro/ for English and Romanian.

output of LexPar, the linkage of the sentence "*The King of Rock & Roll died on August 16, 1977*" is depicted in Figure 1 (punctuation is not included in the linkage):

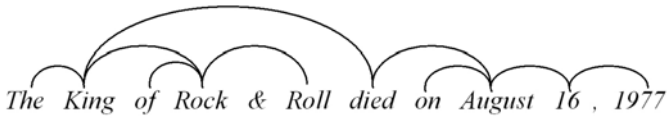

Fig. 1. The result of LexPar on one sentence.

In this figure, we can see that the linker does not follow the usual dependency rules in that this analysis makes its own assumptions about the structure of the dependency graph. For instance, we can see that the preposition "*on*" is not linked to the verb "*died*" as it should be but instead, the head of the noun phrase "*August*" is the one directly linked to the verb. This is because the linker computes the scores for the content words links using association scores so as to be able to detect correct pairings and uses rules of pairing for links involving functional words so that these kinds of links are consistent over all analyses[5]. For example, a determiner will always be attached to a noun subject to agreement rules, other attachments being forbidden (for a detailed explanation of a LAM the reader is referred to [12, 4]).

3 Document Indexing and Searching

The RACAI QA system uses a C# port of the Apache Lucene full-text searching engine. Lucene is a Java-based open source toolkit for text indexing and searching. It is one of the projects of Apache Jakarta and is licensed under the Apache License.

In Lucene, everything is a document. A Lucene document resembles to a row in a relational database and it supports several fields (rows). The type of index used in Lucene and other full-text searching engines is an "inverted index" – every term in the index is associated with a frequency and is mapped to the documents in which it occurred. The index allows Lucene to quickly locate every document associated with a given set of search terms. An important feature of Lucene is the flexibility of the query syntax. We used only a subset of the types of queries supported by Lucene: term, phrase, field-specific and Boolean queries. Field-specific queries can be used to target specific fields in the document index and Boolean queries are used to group the results of the individual queries.

The score of query Q for document D is based on the document term frequency and inverse document frequency expression:

$$score(Q,D) = coord(Q,D) \cdot qnorm(Q) \cdot \sum_{t \in Q \cap D} tf(t) \cdot idf(t)^2 \cdot norm(t,D) \quad (1)$$

– *coord*(*Q*,*D*) is a factor based on how many of the query terms are found in the specified document (a ratio between the terms found in the document and the number of terms in the query)

[5] When more than one rule is applicable at one given time, association scores are used as in the case of content words to select one pairing.

– *qnorm(q)* is a normalizing factor used to make scores between different queries comparable (it is the sum of the squared weights of each of the query terms and it does not affect document ranking since all document weights are multiplied by the same factor);
– *tf(t in d)* is the term frequency in a given document;
– *idf(t)* is the inverse document frequency;
– *norm(t,d)* is a normalization factor associated to a specific document (it is a value computed at index time using the length of the document and the weight given to each field).

3.1 Indexing

Although the Lucene toolkit comes with several already-made tokenizers, stemmers and stop word filters, we preferred to deploy a custom indexing scheme using our own annotated resources. There were considerable improvements when we used the Romanian tokenizer instead of Lucene's default tokenizer because most of the words with hyphen and the abbreviations were handled in a consistent manner. Usually stemming improves the recall to an IR system. We used lemmatization backed by POS-tagging – in most of the cases lemma disambiguates the part of the speech of a word (in Romanian). Instead of filtering the index terms using a stop words list we used the information from POS-tagging to keep only the content words (nouns, main verbs, adjectives, adverbs and numerals). We used the sentence and chunk annotation to insert phrase boundaries into our term index; a phrase query cannot match across different chunks or sentences.

In our implementation every document has different fields for the surface form of the words and their corresponding lemmas. This kind of distinction applies to titles and document text resulting in four different index fields: title word form (`title`), title lemma (`ltitle`), document word form (`text`) and document lemma (`ltext`).

Table 1. Example of indexed terms for the sentence "*Din originea lingvistică se poate observa caracterul istoric al ideii de logică*" ("*From the linguistic origin one can notice a historical feature for the concept of logic.*").

Word	Lemma	POS	Chunk	Word form field	Lemma field
Din	Din	Spsa	Pp#1		
originea	origine	Ncfsry	Pp#1,Np#1	din originea lingvistică	origine lingvistic
lingvistică	lingvistic	Afpfsrn	Pp#1,Np#1,Ap#1		
se	Sine	Px3--a--------w	Vp#1		
poate	Putea	Vmip3s	Vp#1		
observa	observa	Vmnp	Vp#1	se poate observa	putea observa
caracterul	caracter	Ncmsry	Np#2		
istoric	Istoric	Afpms-n	Np#2,Ap#2		
al	Al	Tsms	Np#2	caracterul istoric al ideii	caracter istoric idee
ideii	Idée	Ncfsoy	Np#2		
de	De	Spsa	Pp#2		
logică	Logică	Ncfsrn	Pp#2,Np#3	de logică	logică
.	.	PERIOD			

3.2 Searching

To acquire better precision in ranking the documents and their content we used two indexes: (i) one for the documents (43486 documents, 694467 terms) and (ii) one for the sections of the documents (90819 sections, 700651 terms).

Given a Boolean query with several conjunctive clauses, the system will first try to match all of the query clauses against the document index. If the search does not have a result, the system will recursively try to match n - 1 of the conjunctive clauses until the query returns at least one result from the document index. The returned documents from the conjunctive Boolean query are used to select the corresponding sections in which the query terms occur. The sections are ranked using a query with clauses of the maximal conjunctive Boolean. The clauses from this new query are joined with the disjunctive operator. The ranking is made against the sections index. For instance, for the query in Figure 2, in the first step the system tries to match all the 26 query clauses, but after a few recursive iterations, the maximal query has only 15 matches. The sections from the matched documents are ranked using the maximal query, this time using the operation of disjunction.

Original query:	ltitle:"serial Twin Peaks" AND ltext:"serial Twin Peaks" AND **ltitle:"Twin Peaks"** AND **ltext:"Twin Peaks"** AND ltitle:"serial Twin" AND ltext:"serial Twin" AND **ltitle:Peaks** AND **ltext:Peaks** AND **ltext:crea** AND ltitle:serial AND **ltext:serial** AND **ltitle:Twin** AND **ltext:Twin** AND title:"serialul twin peaks" AND text:"serialul twin peaks" AND **title:"twin peaks"** AND **text:"twin peaks"** AND title:"serialul twin" AND text:"serialul twin" AND **title:peaks** AND **text:peaks** AND text:creat AND title:serialul AND **text:serialul** AND **title:twin** AND **text:twin**
Maximal query:	**ltitle:"Twin Peaks"** AND **ltext:"Twin Peaks"** AND **ltitle:Peaks** AND **ltext:Peaks** AND **ltext:crea** AND **ltext:serial** AND **ltitle:Twin** AND **ltext:Twin** AND **title:"twin peaks"** AND **text:"twin peaks"** AND **title:peaks** AND **text:peaks** AND **text:serialul** AND **title:twin** AND **text:twin**
Results:	

Fig. 1. Example of a query.

4 QA System A

In the IR phase, this system generates queries for every question given which are used to interrogate Lucene. As previously described, for each question the search engine returns a list of snippets in a descent order according to their relevance regarding the provided query. In the IE stage, we first identify the type of the answer required by each question using a Maximum Entropy approach. Based on the answer types and the document and paragraph scores provided by Lucene, the most probable answers are then extracted in a manner that will be described below.

4.1 Query Generation

As previously mentioned, in order to interrogate the search engine for relevant snippets we need to construct a query for each question. The query directly influences the precision and the recall of the engine. Although we primarily aim for a high recall, a good precision will decisively influence the IE phase, as the top outputs are those with highest matching scores. For the current problem, we considered the *recall* to be the percentage of the questions for which the engine returns good snippets (snippets that contain correct answers), from the total number of questions that can be answered, and the *precision* to be the proportion of good snippets from those that have been returned as first ranked.

For generating the queries, the A System uses the content words found in the questions, the noun phrases formed by them and all the subparts of the noun phrases that start with a content word. All these are searched in lemma and word forms both in the title and the text fields of the indexed documents. The query is finally obtained by concatenated all the terms with the logical operator AND (the whitespace separator between terms has AND semantics). As an example, for the question "*Cu ce identifică panteismul divinitatea?*" ("*What does Pantheism identify divinity with?*"), the generated query is:

```
ltitle:"panteism divinitate" ltext:"panteism divinitate"
ltext:identifica ltitle:divinitate ltext:divinitate ltitle:panteism
ltext:panteism title:"panteismul divinitatea" text:"panteismul
divinitatea" text:identifică title:divinitatea text:divinitatea
title:panteismul text:panteismul
```

As mentioned in section 3, the search engine was programmed to return the snippets that contain the majority of the terms. One should know that in the process of query generation we did not take into account those verbs which can be auxiliaries (ex.: fi – be, avea – have). Also, we have to mention that all questions had been previously tokenized, tagged and chunked. The following picture shows the output of Lucene for the previously generated query.

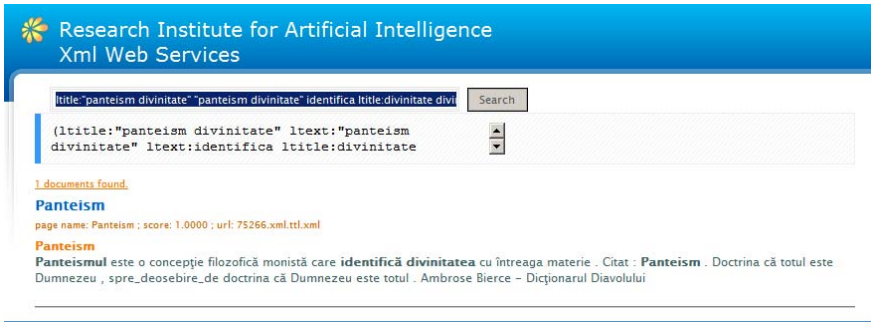

Fig. 2. Lucene output for the example query.

The queries should have been enriched with synonyms extracted from the WordNet lexical ontology, but our tests revealed that, for the given questions, this did not improve the results. This seemed very strange, but we discovered that the reason for this lies in the fact that the answers contained almost all the content words found in the questions. So, expanding the queries with synonyms led to poorer results as a consequence of the noise introduced in this way.

4.2 Question Type Classification

For each question, it is necessary to identify the type of the answer one should search for, meaning that a question needs to be classified with respect to its answer type. In order to recognize the type of the answer, we used a Maximum Entropy approach. Extensively used in NLP for different problems like sentence boundary detection, tagging or text categorization [9], the Maximum Entropy framework is well suited for our task since it can combine diverse forms of contextual information in a principled manner. It is a machine learning technique, originally developed for statistical physics, used for the analysis of the available information in order to determine a unique probability distribution. Similar to any statistical modeling method, it relies on empirical data sample that gives, for some known sets of input, a certain output. The sample is analyzed and a model is created, containing all the rules that could be inferred from the data. The model is then used to predict the output, when supplied with new sets of input that are not in the sample data. The maximum entropy method constructs a model that takes into account all the facts available in the sample data but otherwise preserves as much uncertainty as possible.

Our problem may be formulated as any classification problem: finding the distribution probability p, such that $p(a|b)$ is the probability of class a given the context b. In our case, the context b is formed by certain features extracted for every question. We took into account features like: the first WH word (*cine* - who, *unde* - where, *când* - when, *care* - which, *ce* - what, *cum* - how, *cât* - how many), the existence of other words before the WH word, the existence of certain verbs at the start of the sentence (like *numi* – name), the existence of a word denoting measurement units, the existence of a word denoting temporal units, the number of the first noun, the part of speech of the first content word (noun, verb or numeral), the

existence of the verb "to be" as the first verb, the existence of at least two non-auxiliary verbs, the existence of a proper noun as the first noun or not, the punctuation mark at the end of the question if different from question mark. For the question given as an example above, the features extracted are: "*ce*", "*firstVerbNonDef*" and "*n*" while for a question like "*Numiți trei autori americani ai căror opere au fost influențate de filozofia existențială."* ("*Name three American writers whose work was influenced by the existential philosophy.*"), the features would be: "*firstIsVerb*", "*numi*", "*Number*", "*firstNounIsPlural*", "*firstVerbNonDef*", "*multiVerb*" and ".".

We manually identified the type of answers for 500 questions, and each of the answer type, along with the features extracted, was used for training. It is clear that we did not had enough examples in order to reliably specify $p(a|b)$ for all (a, b) pairs and we needed a method which makes use of the partial evidence to find the best estimation for the distribution p. We took into consideration 8 classes meaning that we might identify 8 types of possible answers: temporal (TMP), time interval (ITMP), definition (DEF), measure (MES), list (LST), location (LOC), names (N) and explanation (WHY). As Ratnaparki showed in [9], this is the typical situation when a ME approach can be employed. The literature describes in detail the mathematical and statistical principles behind this method and we will not further detail them here.

The classifier was tested only on the training data and the questions given for the competition, and, for latter data, its precision was 99.5%, which means that 199 questions from 200 were correctly classified with respect to their type answer.

4.3 Answering the Questions

The determination of the right type of the answer is very helpful in adopting a correct strategy for finding it in the set of snippets returned by the search engine. Among the types we took into account, we identified two as the most problematic: the definitions and the lists. Due to the small amount of time, the A System focused on finding the answers only for those questions that were categorized as requiring definitions as answers. Usually the answer of a question is only several words length but in the case of a definition, a whole sentence may be what we are looking for. The document and section scores provided by Lucene are very important as they provide the strongest evidence regarding the existence of possible answers in the returned snippets. Another important role in finding the answer, no matter the type of the questions, is played by the focus of the question. For the DEF questions in Romanian, we noticed that, usually, the focus of the question is the first NP found in the question starting with a common noun, a proper noun, or an adjective. This is the reason for which we chose the first NP that has those properties as the focus of the question. We discriminated between cases in which the NP has one or more words.

We searched the word form or the lemma form of the focus in every sentence of the sections of the documents returned by Lucene and we looked for several positive or negative clues:

- the existence of "to be" verb (together with a possible auxiliary) immediately following the focus and the existence or not of indefinite articles or demonstrative pronouns or articles (*cel* in *cel care* – the one who or *cel mai* – the most) after the verb; (positive)

- the existence of an opened left bracket immediately after the focus, followed or not by a noun; (positive)
- the existence of a comma in front or after the focus; (positive)
- the existence of certain prepositions before the focus (*la* – at, *pentru* - for); (negative)
- the existence of a definite oblique article in front of the focus; (negative)

Usually in a document the definitions are given in the beginning so we expect that the first sentences of the documents contain them. This is why we penalize the candidates as we find them farther and farther in the document (the length in sentences counts) (sentence_score = 1.0 – current_sentence_index * 1.0 / number_of_sentences * 2). The rank of a candidate is taken into account, no matter the document, or the section (last_added_score = 1.0 – rank_of_candidate * 0.015). The formula also includes, as previously stated, the document and section scores for the snippets in which the candidates were found (total_score_for_a_candidate = doc_score * section_score * sentence_score * last_added_score). Finding the focus only in lemma form or finding only parts of the focus will penalize the newly added candidates (the candidate score is weighted with 0.75, respectively 0.6). Different combinations of the positive clues led to the weighting of the total score with values between 0.6 and 1.5. All the weights were empirically set by tuning the system from the previous CLEF competition.

5 QA System B

This QA system is the second of the two QA systems that were combined in order to obtain the final result. This system uses the linkage of the question to determine the question focus and topic and their dependants and also to generate the formal query to the text searching engine. In what follows, we will describe how the question is analyzed (i.e. focus/topic identification, query expansion) and how the answer is extracted from the text passages returned by the text searching engine.

5.1 Question Analysis

The input question is first preprocessed with TTL to obtain word tokenization, part of speech tagging and lemmatization and then, it is analyzed using LexPar. The linkage of the sentence is used to extract the focus-topic articulation of the question along a pattern, which is a sequence of parts of speech belonging to words that are directly linked in the question. Here are the most important patterns for Romanian (considered from the beginning of the question):

1 `[preposition], {WH determiner}, {noun(FOCUS)}, {main verb}, {noun(TOPIC)}` i.e. "*În(*English *In)*/**preposition** *ce(what)*/**WH determiner** *an(year)*/**noun** *s-a născut(been born)*/**main verb** *Mihai(Mihai)*/**noun** *Eminescu?*";

2 `{WH pronoun(FOCUS)}, {main verb}, {noun(TOPIC)}` i.e. "*Cine(Who)*/**WH** **pronoun** *este(is)*/**main** **verb** *Mihai(Mihai)*/**noun** *Eminescu?*";

3 `{WH adverb(FOCUS)}, {main verb}, {noun(TOPIC)}` i.e. "*Unde(Where)*/**WH** **adverb** *se află(is)*/**main** **verb** *lacul(the lake)*/**noun** *Baikal?*";

4 `{main verb}, {noun(FOCUS)}` i.e. "*Numiți(Name)*/**main** **verb** *culorile(the colors)*/**noun** *steagului României.*".

After the focus and topic are extracted, the query to the Lucene full-text searching engine is created by following the links in the linkage of the question so as to extract all the links that are formed between content words (nouns, main verbs, adjectives and adverbs). With this list of links at hand, the query is computed as a logical disjunction of terms in which each term corresponds to a content word to content word link and it is equal to a logical conjunction of the lemmas at the end points of the link. For instance, for the question "*În*/**prep** *ce*/**wh-det** *an*/**noun** *s-*/**refl-pron** *a*/**v-aux** *născut*/**v-part** *Mihai*/**noun** *Eminescu*/**noun**?" with the linkage {<*În*, *an*>, <*ce*, *an*>, <*an*, *născut*>, <*s-*, *născut*>, <*a*, *născut*>, <*născut*. *Mihai*>, <*Mihai*, *Eminescu*>}, the list of the links between content words is {<*an*, *născut*>, <*născut*. *Mihai*>, <*Mihai*, *Eminescu*>} and the resulting query is:

```
(ltext:an AND ltext:na•te) OR (ltext:na•te AND ltext:Mihai) OR
(ltext:Mihai AND ltext:Eminescu)
```

As previously explained, our text searching engine does not use this query directly but it heuristically tries to replace the OR operators with AND operators until it gets one or more hits (in other words, it tries to refine the initial query until it finds one or more documents that satisfy the refined query). If replacing doesn't help, the initial query is tried and the results are returned to the QA application.

5.2 Answer Extraction

Answer extraction is basically the best structural match between the linkage of the question and the linkages of the sentences in the paragraphs that have been returned by the text searching engine in response to the automatically formulated query (see previous section).

Because the linkage is not a full dependency parse, we do not know which word in the sentence is the root word of the dependency tree. We solve this problem by choosing the first main verb in the sentence/question to be the root of the linkage. To better explain how structural scoring is made between the linkage of the question and the linkage of one document sentence, is best to see an example (see Figure 3).

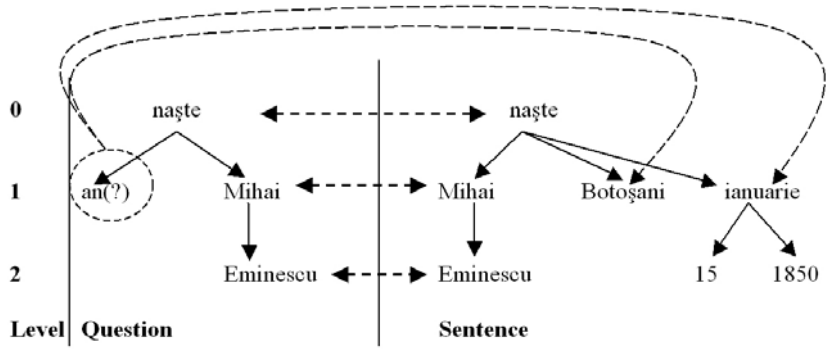

Fig. 3. Structural match between the question "*În ce an s-a născut Mihai Eminescu*" and one candidate sentence.

To follow the same example, for the question "*În ce an s-a născut Mihai Eminescu?*", the text searching engine returns among other paragraphs, one which begins with the sentence "***Mihai Eminescu*** *s-a* ***născut*** *la Botoşani la 15 ianuarie 1850.*" (the keywords from the question are bolded). In Figure 3, on the left side we have the linkage of the question (functional words removed) and on the right, we have the linkage of the candidate sentence (functional words also removed). Structural match means going depth-first through the question tree one node at the time and for each such node (let it be *Nq*), going depth-first through the sentence tree searching from the current node in the question tree (let *Ns* be the matching node). When such a node is found, a matching score *S* is increased by $1/(1 + |depth(Nq) - depth(Ns)|)$ such that if the nodes are at the same depth in the two trees, the value of *S* increases by 1. Otherwise, the value of *S* increases by the inverse absolute difference of depths at which matching nodes are found. For the two trees in Figure 3, $S = 3$ (see the dotted arrows which mark 3 matching nodes at the same depths).

After the structural match score *S* is computed, we extract all the subtrees from the candidate sentence tree such that: a) subtrees do not contain already matched nodes and b) they are at the same depth as the focus node of the question (marked with the "?" in Figure 3). For our example, we have two such subtrees: "*Botoşani*" and "*15 ianuarie 1850*", which, in the absence of a proper named entity recognition ("*Botoşani*" is the city of birth of Mihai Eminescu and "*15 ianuarie 1850*" is his full date of birth), are equal answer candidates. To solve this problem and remembering that the POS tags are the only indications as to the type of entities we are dealing with, we have assigned to each focus-topic extraction pattern a POS indicating the type of answer we are searching. In our example, this question pattern has a numeral POS attached, so we choose the answer that contains numeral(s), namely "*15 ianuarie 1850*" which is a bit longer than the exact answer: *1850*.

Structural match occurs between the linkage of the question the linkage of each sentence of each paragraph that was returned by the text searching engine. We want to order the candidate sentences by the *S* score but also by the score of the paragraph in which the sentence occurred (we want to also give credit to the text searching engine).

This way, the final candidate sentence score is $A = \alpha S + (1 - \alpha)P$ where P is the score of the paragraph containing the candidate sentence[6]. Answers are thus extracted (as explained above) from the top candidate sentences ordered by the A score.

5 Results and Conclusions

The results obtained should be analyzed considering both Information Retrieval and Information Extraction phases. As in the IE stage we used the output of the IR when looking for the correct and exact answers, it is clear that the two systems (A and B) could not answer a number of questions greater than that for which the search engine returned good snippets. Besides the manner the documents were indexed and the engine's parameters tweaked, its results depend on the way the queries are constructed. Although the queries were built in two different ways, as described in the sections about the two systems, it turns out that the outputs obtained were very similar. The experiments showed that we got somehow improved results with the query generated by the A System. In most of the cases, using these queries meant finding the right snippet higher in the search engine's list of the returned snippets. This lead to an increased precision, the difference between the precisions of the two systems being around 4%. However, in very few cases, when the right snippet was hard to find and the queries obtained using the A Systems' method did not get it at all, the queries produced with the B Systems' technique made the search engine return it, but only in the end of the findings list. So, we got a 2% higher recall for the B System. The actual figures for the two systems were manually computed as described in 4.1 section and they are shown in the table below.

Table 2. The evaluation of the IR phase for both of the systems.

	Precision	**Recall**	**F-measure**
A System	68% (136 questions)	77.72% (150 qs)	72.53%
B System	64% (128 qs)	79.79% (154 qs)	71.02%

We should mention that some of the questions given to be answered were related. These questions were arranged into groups, for most of the questions, to retrieve relevant results we had to take into account information found in the previous questions. We managed to handle this situation by adding the query generated for a new question to the query of the first question of the group.

Due to the small amount of time, in the IE phase we used the strategy of splitting the tasks between the two systems. The A System focused on answering the DEF type questions, while the B System concentrated on answering the other questions. However, the B System did not use the information provided by the classifier described in the 4.2 section in searching for the correct answers.

From the total of 200 questions, 31 were identified as requiring a DEF type answer but only 30 were in fact of this nature. The A System answered correctly with the first

[6] Actually, because $0 < P \leq 1$ and $S > 1$, we replaced P by $10 \times P$ to get P in the same range as S. With this setting, the best experimental value of α is 0.4.

returned output in 25 cases although in 3 situations the answers were considered inexact (possibly because of their length). However, for another 4 questions one could have found the correct answer among the first three replies.

For the question "*Ce este Selena?*" / "*What is Selene?*", the top 3 answers returned by the system were:

1. "*o actriță şi cântăreață americană , născută pe 24 iulie 1969 , în cartierul Bronx din New York* ." / "*an American actress and singer, born on July 24, 1969 in Bronx, New York* .";
2. "*satelitul natural al Pământului* ." / "*the natural satellite of the Earth*";
3. "*cauza mareelor şi a modificat continuu durata mişcării de rotație* ." / "*the cause of the tidal and continually modified the length of the rotational motion*".

The second answer was considered the correct one, although we believe that the first one is also good enough. Obviously, the third answer is wrong since it's not even sufficiently coherent.

The answers extracted by the B System for the non-DEF type questions, were handled in two ways, which correspond to the two runs we sent to the organizers. First we considered the first answer as the good one. The second approach was to construct the good answer by concatenating the majority of the answers coming from the same snippet. In this way we managed to transform 4 wrong answers into inexact ones, as shown in the table below. However, the total score remained unchanged.

Table 3. The official results.

	First Run	**Second Run**
Total	- 60 Right **- 105 Wrong** **- 34 ineXact** - 1 Unsupported **Overall accuracy = 60/200 = 30.00%**	- 60 Right **- 101 Wrong** **- 39 ineXact** - 0 Unsupported **Overall accuracy = 60/200 = 30.00%**
Factoids	Total Factoids: 160 Right: 38; Wrong: 90 Unsupported: 1; InExact: 31 Accuracy = 38/160 = 23.75%	Total Factoids: 160 Right: 38; Wrong: 86 Unsupported: 0; InExact: 36 Accuracy = 38/160 = 23.75%
Lists	Total List Questions: 10 Right: 0; Wrong: 10 Unsupported: 0; InExact: 0 Accuracy = 0/10 = 0.00%	Total List Questions: 10 Right: 0; Wrong: 10 Unsupported: 0; InExact: 0 Accuracy = 0/10 = 0.00%
Definition Questions	Total Definition Questions: 30 Right: 22; Wrong: 5 Unsupported: 0; InExact: 3 Accuracy = 22/30 = 73.33%	Total Definition Questions: 30 Right: 22; Wrong: 5 Unsupported: 0; InExact: 3 Accuracy = 22/30 = 73.33%
Temporally Restricted Questions	Total Temp. Restricted Questions: 51 Right: 10; Wrong: 31 Unsupported: 0; InExact: 10 Accuracy = 10/51 = 19.61%	Total Temp. Restricted Questions: 51 Right: 10; Wrong: 31 Unsupported: 0; InExact: 10 Accuracy Questions = 10/51 = 19.61%
NIL Answers Returned	Total NIL Returned: 54 Right: 7; Wrong: 47 Unsupported: 0; InExact: 0 Accuracy = 7/54 = 12.96%	Total NIL Returned: 54 Right: 7; Wrong: 47 Unsupported: 0; InExact: 0 Accuracy = 7/54 = 12.96%

We should notice that we did not answer correctly to any of the *Lists* questions because there was no separated strategy in this direction. The improvements can definitely come from using the question classifier and from adopting specific strategies in searching for different types of answers.

References

1. Gaizauskas, R., Humphreys, K.: A Combined IR/NLP Approach to Question Answering Against Large Text Collections. In: 6th Content-Based Multimedia Information Access Conference (RIAO-2000), pp. 1288–1304. Paris, France (2000)
2. Harabagiu, S., Moldovan, D., Clark, C., Bowden, M., Hickl, A., Wang, P.: Employing Two Question Answering Systems in TREC-2005. In: Text Retrieval Conference (TREC-14). Gaithersburg, Maryland (2005)
3. Harabagiu, S., Moldovan, D., Paşca, M., Mihalcea, R., Surdeanu, M., Bunescu, R., Gîrju, R., Rus, V., Morărescu, P.: FALCON: Boosting Knowledge for Answer Engines. In: Text Retrieval Conference (TREC-9), pp. 479—489. Gaithersburg, Maryland (2000)
4. Ion, R.: Word Sense Disambiguation Methods Applied to English and Romanian. PhD thesis, Romanian Academy, Bucharest (2007)
5. Ion, R., Barbu Mititelu, V.: Constrained Lexical Attraction Models. In: Nineteenth International Florida Artificial Intelligence Research Society Conference, pp. 297–302. AAAI Press, Menlo Park, Calif., USA (2006)
6. Moldovan, D., Bowden, M., Tatu, M.: A Temporally-Enhanced PowerAnswer in TREC 2006. In: Text Retrieval Conference (TREC-15). Gaithersburg, Maryland (2006)
7. Moldovan, D.I., Clark, C., Harabagiu, S.M., Hodges, D.: COGEX: A semantically and contextually enriched logic prover for question answering. J. Applied Logic 5(1): 49-69 (2007)
8. Moldovan, D., Harabagiu, S., Paşca, M., Mihalcea, R., Goodrum, R., Gîrju, R., Rus, V.: Lasso: A Tool for Surfing the Answer Net. In: Text Retrieval Conference (TREC-8), pp. 175—184. Gaithersburg, Maryland (1999)
9. Ratnaparkhi, A.: Maximum Entropy Models for Natural Language Ambiguity Resolution. PhD thesis, University of Pennsylvania, Philadelphia, PA (1998)
10. Voorhees, E.M.: Overview of the TREC 2005 Question Answering Track. In: Text Retrieval Conference (TREC-14). Gaithersburg, Maryland (2005)
11. Waldinger, R.J., Appelt, D.E., Dungan, J.L., Fry, J., Hobbs, J.R., Israel, D.J., Jarvis, P., Martin, D., Riehemann, S., Stickel, M.E., Tyson, M.: Deductive Question Answering from Multiple Resources. In: New Directions in Question Answering 2004, pp. 253—262. (2004)
12. Yuret, D.: Discovery of linguistic relations using lexical attraction. PhD thesis, MIT, Cambridge, Massachusetts (1998)