Adapting Question Answering Techniques to the Web

Jignashu Parikh
Intelligent Systems Lab,
Computer Science and Automation Department,
Indian Institute of Science,
Bangalore, India.
jignashu@csa.iisc.ernet.in

M. Narasimha Murty
Intelligent Systems Lab,
Computer Science and Automation Department,
Indian Institute of Science,
Bangalore, India.
mnm@csa.iisc.ernet.in

Abstract

The Web has emerged as a huge information repository that can be used for various knowledge based applications, an important one being Question Answering (QA). The paper discusses the issues involved in using web as a knowledge base for question answering involving simple factual questions. It also proposes some simple but effective methods to adapt traditional QA methods to deal with these issues and lead to an accurate and efficient Question Answering system. The paper discusses the architecture and design of our system. The empirical results obtained on the current system are encouraging.

1 Introduction

Question Answering (QA) has been studied in Natural Language Processing (NLP) since 1970s with the systems like LUNAR [10] which allowed geologists to ask questions about moon rocks. QA is defined to be an Information Retrieval (IR) paradigm for discovering answers to an open-domain natural language question from a large collection of texts. Traditionally QA systems have been applied to fixed repositories of text documents.

While talking of document repositories one cannot skip thinking about the World Wide Web. As on 1st October, 2002, Google [8] has indexed about 2.5 billion documents and it is ever increasing. The best thing about WWW is that the information there is across various domains and is kept up-to-date with the web itself. These factors make WWW a very useful knowledge base for general purpose QA systems. But with the size of the web comes the difficulty of pin-pointing the *desired* information. Search engines like Google do solve this problem to a certain level in terms of suggesting the *right* documents but when a user requires a one line answer to his question, a natural question is *why sift through a huge pile of data while looking for a one line answer?* This exactly is the problem that we focus on in this paper.

In this paper, we focus on question answering for simple factual questions that require brief (typically one line) answers, also referred to as *factoids* by Hovy (2002). User expects very brief answers to questions like *Who was the first person to climb Everest? Where is Taj Mahal located? etc.* Fine grained information extraction techniques need to be used for pinpointing answers within *likely* documents. Most of the QA systems have a similar framework: they generate a query out of the natural language question, perform document level IR and then pinpoint the exact answer. Systems vary mostly in the first and third phases of query generation and pinpointing answers. In our system, we have tried to adapt QA techniques for the web. The reason why we need to do this are:

1. The content on the WWW is not monitored. Correct formatting and grammar are not ensured.

   Apart from this, dealing with falsehoods is a major difficulty for QA systems. For example, for the question *Who invented the electric guitar?* One of the websites gave the answer *The electric guitar was invented by Michael Jackson in 1875 and was candle powered.* And later it said that it was a joke.

2. Web pages are constantly updated. Information found in a page when a search engine indexed it, may not be there when the system refers to that page on a user query. The new page might be irrelevant for the query now. It is unrealistic to assume that any search engine will ever be able to keep its index up-to-date always.

3. The queries are to be executed over the internet and cannot therefore afford delays due to very involved processing. The system should come up with

Proceedings of the Language Engineering Conference (LEC'02)
0-7695-1885-0/02 $17.00 © 2002 IEEE

accurate answers in the simplest and fastest possible ways.

Often traditional systems use semantic processing using WordNet [5], we explain in section 2 why these do not work and in fact might even reduce accuracy.

We propose a QA system in this paper that deals with the above issues and achieves accuracy with simple computations. The design decisions taken for these are discussed in the next section. The complete system architecture and the details of its working are presented in sections 3 and 4 respectively. In section 5 we compare our system with various other QA systems. We give some interesting empirical results in section 6. The future work and conclusion follow in section 7.

2 Design Issues

There are certain design decisions that are at the heart of the system. These decisions are taken to deal with the issues discussed in the previous sections.

The lessons learnt from Google's design in document level IR related issues have influenced our design decisions. The major design considerations are as follows:

1. Google has always believed in *Fast is better than slow*. Following this, we have taken many decisions as given below:

   a. The system should try to present answers in a format understandable by the user but not necessarily in the exact grammatical form the answer should be. For example, for questions like,

      Where is the Gateway of India located?

      The right way to answer this question is to say:

      The Gateway of India is located in Mumbai.

      However, even if we present the user with a sentence like

      The Gateway of India, located in Mumbai, is one of the most visited places in India.

      The user can obviously make out the answer present in the above sentence and we feel that rigorous analysis of sentences like the above and converting them into grammatically *right* versions of answers is not required. Not only it saves processing but also eliminates errors that may emanate from incorrect analyses. Systems like MUDLER [2] try to generate one word answers but for this they consider all noun phrases with the answer sentence that they extract as candidate answers, which obviously may lead to incorrect candidates, although many of them are eliminated through the voting procedure.

      Thus, our system should help the user in pinpointing the relevant information within documents but expects a bit of *common sense* from the user's part in getting the answer from the answer sentences that we present.

   b. Google does not perform any stemming of words. This is one of the reasons for accuracy in search results. For instance, once you change a query word, say *reader,* to *read* by stemming the search results may be very different than what the user expects. Similarly, in our system we do not attempt to stem any word. Also, systems like Webclopedia [9] propose expanding queries by replacing query words with its synonyms using synsets from WordNet in order to boost recall. We believe that this can be dangerous since while boosting recall precision may suffer badly. There are several reasons why this can happen but the main reason is for every word, WordNet presents multiple synsets, each representing a different sense of the word. How do we know which synset to select for replacing the word with its synonyms? For instance, replacing the word *pipe* with its WordNet synonym *tube* (sense 3 of *pipe*) when the user intends *smoking pipe* (sense 1) may give irrelevant results and thus may affect precision badly. To get it right we need a good word sense disambiguation system but sense disambiguation is known to be an AI-complete problem. Mark Sanderson concludes in Sanderson (1994) that performance of IR systems is insensitive to resolution of ambiguity but very sensitive to erroneous disambiguation. This supports our decision to avoid any kind of stemming, ambiguity resolution or word replacement.

2. Another sensitive issue is to deal with uncertainties with regard to our knowledge base-the web. There are various problems with regard to this:

   - The server hosting site that has been referred to by our search engine for answering a question may be down or very slow.

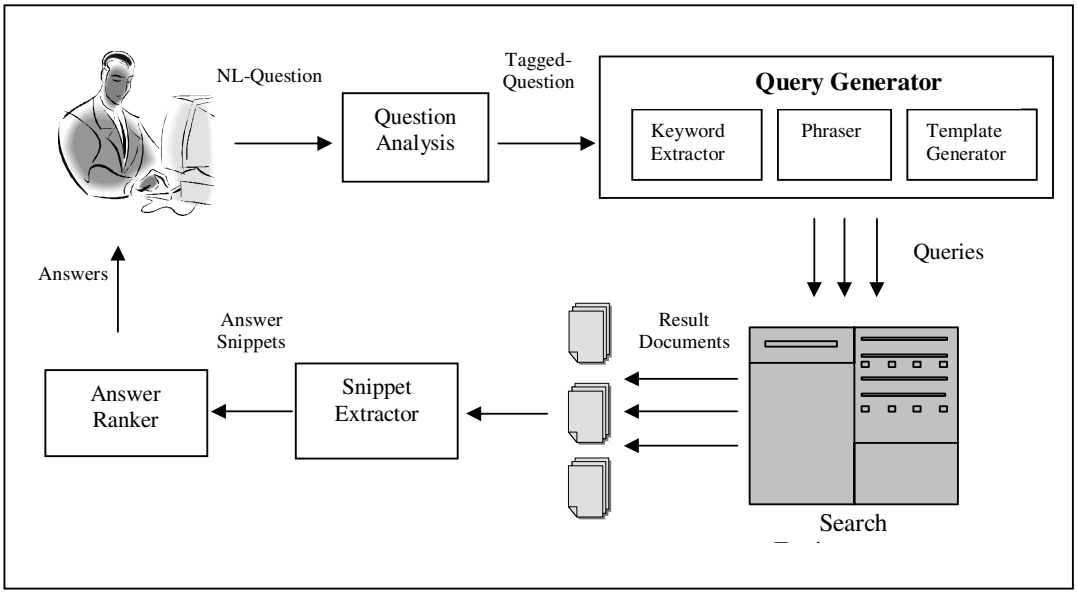

Figure 1: Architecture of the System

- The page might be removed from the server. The page might even be updated in a way that it might not be able to answer our question anymore.

These issues can have a lot of impact on system performance. As we are analysing multiple pages for answer extraction the speed our system will be limited by the slowest of them. To deal with this, we suggest a very simple solution which is to refer to the search engine's cached pages instead of looking at the actual pages. Although the information on the cached page might be slightly out dated but *it will be there*! Also we can now depend on powerful servers of search engines like Google and get uniform speed access for all pages. This solution is very much acceptable for a general purpose QA system like ours but for a time-critical system like QA on stock quotes, obviously we need to come up with better solutions.

3. We focus on improving precision rather than recall. For factual questions, it takes one correct page to answer a question. To incite confidence we might present a small number of pages (say 10 or 20) so that the user can see what the majority of web pages say. But if our precision is bad, the user might get misguided or might not get any relevant answers at all. Hence, we try to get a small number of *right* answers rather than get *all* of them.
4. Use parallel processing whenever possible. The response times achieved by Google are mainly due to the large number of resources employed and run in parallel. Our system is designed to be a multi-server and multi-threaded wherein every server fetches answer documents for a particular query and every thread on that server analyses one document out of those documents.

These points summarize some major design decisions for our system. The system itself is discussed in the next section.

3 Overview of the System

Figure 1 shows the architecture of our system. The system takes in a natural language (NL) question in English from the user. This question is then passed to a Part-of-Speech (POS) tagger which parses the question and identifies POS of every word involved in the question. This tagged question is then used by the query generators which generate different types of queries, which can be passed to a search engine. These queries are then executed by a search engine in parallel.

The search engine provides the documents which are likely to have the answers we are looking for. These

documents are checked for this by the answer extractor. Snippet Extractor extracts snippets which contain the query phrases/words from the documents. These snippets are passed to the ranker which sorts them according to the ranking algorithm as described in the next section.

4 Design of the system

This section describes various components of our system. It also gives a detailed view of the working of the system right from the point a query is given to the system to the output of the answers.

4.1 Question Analysis Module

The natural language query given by the user is first tagged by a part of speech tagger. For our implementation we have used QTag tagger [7] which is known to accurately and efficiently tag sentences. This module generates a POS tagged form of the input sentence which is used by the query generation module.

4.2 Query Generation Module

This module performs the important task of generating various queries to be fired to the search engine. This module analyses the POS-tagged sentence and generates queries that can be passed to a search engine. These queries are of various levels of constraints. For example, the most general query that can be extracted from a NL-query is a keyword query comprising of important words: nouns, adjectives, verbs and adverbs. The different types of query generators present in the system are described below. We will use a running example of the question *Who was the irst president of India?* to explain each of them.

a. **Answer Template Generator:** Given a question, we generate possible answer templates that are likely to produce the best answers to the given question. These are passed to search engine as phrase queries (in quotes).

The system has a transformation grammar [1] module that generates such templates for a specific question template. For instance, if the question template is:

Who [auxilliary-verb] [Noun Phrase] ?

The corresponding answer template will be:

"[Noun Phrase] [auxilliary-verb]"

For example, answer template corresponding to our example question is:

"The first president of India was"

OR

"was the first president of India"

The utility of generating queries like the above is very straightforward. When passed to Google (along with the quotes), this query will fetch all pages which surely has the above phrase and it is highly probable that the object of the sentence which contains the phrase above is a proper noun- the name of the president.

Other possible pattern that may suggest an answer to the above question is:

", the first president of India, "

This when put into a sentence may look like:

*"Dr. Rajendra Prasad**, the first president of India,** was born in 1884"*

Any reader going through the above sentence easily gets the answer he wanted, viz., the name of the first president of India.

At this point, one may ask that there may not be any pages on the web that correspond to the answer templates generated. But we believe that given the current size of the web and that we issue multiple answer templates as generated above, it is highly probable that at least one of these will succeed. In case they do not, there are more flexible patterns generated which are described next.

b. **Phraser:** The phraser module generates composite nouns and noun phrases. This module is at a middle tier in terms of flexibility between keyword and answer template generators. At the same time the probability of correct answers possible from queries generated by this module are more than those from keyword generator but less than those based on answer templates.

The processing that happens in this module is quite simple. It has a predefined grammar for identifying noun phrases and compound nouns. The rules in this grammar are of the form:

NP → ARTICLE NOUN
→ ARTICLE ADJECTIVE NOUN
→ ARTICLE ADJECTIVE NOUN "of" NOUN
→ ADJECTIVE NOUN NOUN
→ ...

For the example question above, the phraser will generate the following:

"the first president of India"

It retains any verbs present in the question. For example, if the question is:

Who invented the electric guitar?

The corresponding query generated by phraser is:

"invented" "the electric guitar"

Notice that the word *invented* was not changed to its root *invent*. This is again because the desired word for the answer is *invented* and not *invent*. The sentences that might suggest us the answer are likely to have the forms like,

1. *The electric guitar was invented by Les Paul.*
2. *Les Paul invented the electric guitar.*

If *invent* is passed to a search engine like Google it will prefer pages which have *invent* as compared to *invented* and thus leads to a bad query.

c. **Keyword Extractor:** This is amongst the simplest of query generators. This module just picks up the *important* words (keywords) from a given question. This selection is based on the part of speech of the words involved. The heuristic used for this is to select all nouns, adjectives, verbs, adverb and adjectives from the given NL question. Thus, the query for our example question will be:

"first" "president" "India"

This being the most flexible form of query, we can safely say that if this query fails probably the answer is not there on the web. It is surely possible that the answer is there on the web with the synonyms of the keywords used in the question but we do not attempt to try for that because of the reasons given in section 2.

4.3 Search Engine

Search engine is one of the most important parts of our system since our knowledge base is the web. The quality of answers depends on high-quality precise documents. We choose Google for this because of several reasons as follows:

1. It has the most comprehensive coverage and hence means a larger knowledge base for our system.
2. Google is known to give very accurate answers with high efficiency. The credit for this goes to its unrivalled ranking policy.
3. Google's search strategy requires all query words to be necessarily present in every document it retrieves. Also, Google ranks documents with smaller distance between keywords over others. This helps in answer extraction since, as we explain later, we extract answer templates which have all keywords in the same sentences. Google's strategy helps us in doing this.

These advantages of Google make it an ideal choice for our system. However, there is one limitation with Google with respect to what we desire for the system. As we have seen in the previous sub-section that we do a pattern search with answer templates that also include punctuation marks (for e.g., ", the first president of India,"). Google ignores punctuation marks like comma (",") in its search. The system deals with this issue through the Snippet Extractor as discussed in the next section.

4.4 Snippet Extractor

This module extracts answer snippets from the documents returned by the Search Engine. We considered several different techniques but ultimately adopted a way similar to what Google does for web pages. We extract snippets containing lines that have *all* the query words/phrases and a line before and after that line. This is similar to Google's approach of having a default *AND* condition between keywords for retrieving documents. We enforce this condition for extracting snippets and have found that it leads to very high accuracy than allowing lines which either do not have all keywords or have keywords scattered over different lines. For example, consider the following query corresponding to our example question:

"first" "president" "India"

When we allowed these keywords to be scattered across neighbouring lines, we got answer snippets like the following:

*... Clinton was the **first president** to visit the region in 22 years. The last **president** to do so was Jimmy Carter. Pakistan and **India** have ...*

On the contrary, the condition of having all the keywords to be present in a line eliminates such redundant answers.

We select adjacent lines to the given sentence because at times they help in dealing with anaphoric references. For example, we might have a possible answer to a phrase query "the first president of India" can be:

Dr. Rajendra Prasad was born in 1884. He later became the first president of India. He…

Selecting only the center line *He later…* will not give any clue to the reader as to what the actual answer is. It may so happen that the answer is not even in the neighboring lines but it helps the user to get some more clues about the answer. At worst he can refer to the source website itself.

Snippet extractor does a strict checking of query membership in a line with respect to punctuation marks. Google does not take such punctuations into account. This means that even though the query is present in all the documents that Google returns, not all of them will lead to desirable answer snippets and system might need to throw away some of the documents retrieved. This also means more work for the system but improves accuracy.

4.5 Answer Ranker

Question Answering systems typically have a uniform ranking strategy for all types of queries ([2], [9]). However, in our system, we have different ranking strategies for answers generated by various types of queries. Firstly, there is ordering of answers across query types and secondly answers from a particular query type are ranked with different strategies.

Since the answer template queries have highest possibility of getting the right answers the answers generated from them are given the highest rank followed by phrase and keyword queries respectively.

For answer template queries, the ranking mechanism is done manually based on the desirability of the format of answer that will be generated from the answer template. For example, the answer template query "The first president of India was" will generate better sentences than the query ", the first president of India," since the latter gives indirect answers while the former may generate exact answers. Assigning such preference for different answer templates is not very difficult because we will have a small set of answer templates to deal with for a given question template.

For phrase and keyword queries we resort back to the conventional ranking methods that take into account the distance between keywords/phrases and their Inverse Document Frequency (IDF) values. We use the method of ranking based on the one discussed in Kwok (2002).

The ranking function prefers answer snippets that contain more important keywords which are close to each other. To determine the importance of a keyword, we use a common IR metric known as *inverse document frequency* (IDF), which is defined by N/df , where df is the size of a document collection and N is the number of documents containing the word. The idea is that unimportant words, such as *the* and *is*, are in almost all documents while important words occur sparingly. For unknown words, we assume df is 1. To measure how close keywords are to each other, we calculate the square-root-mean of the distances between keywords. For example, suppose $d_1,...,d_{n-1}$ are the distances between the n keywords in snippet s , then the distance is,

$$D(s) = \frac{\sqrt{d_1^2 + ... + d_{n-1}^2}}{n-1}$$

If s has n keywords each with weight (IDF) w_i , then its score is given by,

$$S_s = \frac{\sum_{i=1}^{n} w_i}{D(s)}$$

We do not order answer template queries based on the above metric because all these queries have a single keyword, viz., the answer template and the IDF value of all such keywords will usually be 1.

5 Related Work

We will divide this section into two parts. First we will relate our work to the conventional QA systems and will follow it up with a discussion on contemporary web based systems.

Conventional QA systems can be divided into ones that depend on a knowledge base for answering (e.g. [12], [13]) and ones that are made for text (pre-tagged) corpora (e.g. [14], [0]). Among first such systems were BASEBALL [17] and LUNAR [10]. These systems worked on a structured database. One of the most important works on QA is the START system [13]. However, dependence on the knowledge base limits the scope of questions that such systems can answer. The scope of BASEBALL systems was limited to queries on Baseball, LUNAR on geological information for lunar rocks and START also has a fixed set of web sources that it has converted into a knowledge base. Any such system will face difficulties of creating and managing a knowledge base.

On the other hand, most of the information available is in plain text form. Recently there is an upsurge in the research on developing QA systems on the web. Important ones amongst these are the Mulder [2], Webclopedia [8], Answerbus [11], AskJeeves [20], IONAUT [18] and QuASM [19].

Ask Jeeves is different amongst all of these in a couple of ways. First, it is not an automated system and second, it does not try to extract answers but delivers relevant documents. The system has a large number of questions mapped with corresponding relevant documents (done manually). Given a new question, the system tries to map the question with a similar question from its repository of questions. This again has limitations similar to the systems that depend on knowledge bases. The

scope of the system is limited to its repository of questions.

IONAUT and QuASM both return chunks of data as answers and require the user to search through them. Thus, they are not specifically designed for answering factual questions.

Webclopedia works on a local TREC database. Like our system it returns answers in a single sentence. However, it employs techniques like query expanding by using synonyms of the query keywords. It does this by using WordNet and as explained in section 2 this is bound to reduce precision. Also, the system tries to do quite an amount of semantic processing which increases the response time.

AnswerBus like our system is tuned to work on the web and looks for better response times. However, it ignores the answer template queries that significantly improve the quality of answers. It [11] does an evaluation based on presence of the correct answers in top 1 and top 5 answers that their system returns. However, this doesn't measure how good are the answers extracted by the system. Also, the case of having one correct and more than one false answers is not accounted for in this matrix. On the other hand, the matrix for evaluation chosen by us, viz., precision, stands for quality of answers retrieved or how useful the answers extracted were. As it is shown in section 6, our system performs significantly better than AnswerBus on the precision front.

Systems like Mulder and AskMSR [16] support our usage of answer templates. Brill et. al. explain how redundancy of information stored on the web makes it different from conventional document repositories and this can be used effectively to avoid complex syntactic and semantic processing for QA. However, the answer template generation in the AskMSR is inefficient. For example, if the query is "Where is w_1, w_2, … , w_n?", their system generates multiple templates like "w_1 is w_2 … w_n", "w_1 w_2 is … w_n", etc. without doing any syntactic processing to find out where exactly to place *is*. This not only is in efficient but at times it is not possible to generate answer templates. Consider the question,

When did Abraham Lincoln die?

The answer template for this question will be

"Abraham Lincoln died in"

This requires producing the past tense form of the verb *die* and is not possible unless you know which word is the verb in the sentence. Thus, it calls at least for part of speech tagging.

Mulder attempts to find the exact answer to the question in a singe word. This again requires expensive NL processing of the answers snippets to get it right or else can result into incorrect answers. Giving answers in a single word leaves out contextual information that is important for users to judge whether an answer is a correct one.

6 Empirical Evaluation

This section presents some experimental results of our systems. We first present the current specifications of the system and then compare the system performance with yet another question answering system- *AnswerBus* [11]. We choose this because it is closest to our system in terms of functionality and is available freely on the web.

6.1 System Specifications

The system is currently implemented on a linux cluster of 5 Pentium III machines. Four of these machines act as servers for querying and fetching documents from Google. The fifth machine controls the system and also generates queries to be passed to all the servers. We currently select first 10 results returned by Google. We typically fire about 3 queries per sentence and thus will be analysing about 30 documents. Each of the 10 results obtained per query is analysed by a separate thread in the server.

Currently the system handles keywords and phrase queries for all types of questions. The answer template queries have been implemented only for questions starting from *who*.

6.2 Experimentation

We present performance numbers for all three types of query generators. Answer template queries are only for *who, where and when* type of questions. We have compared our performance with AnswerBus system for TREC-9 questions from the question answering track [3]. The performance numbers are presented by our evaluation from 50 questions taken from TREC 9 question set. The results from both the systems have been manually tested for relevance.

The metric used for comparison is precision because of the reasons we explained in section 2. We define an answer to be relevant if it answers the given question, either correctly or incorrectly. It can be seen like we are asking several experts for an answer and some of them may give you incorrect answers and then we go by the majority of them. An answer which tells you about a related matter is not considered valid. For instance, *The inventor of solid state television is …* is not considered a valid answer for a question *who invented the television?*

The result based on these 50 questions is given in Table 1. The table presents precision numbers for all the three types of queries and compares the overall precision with that of AnswerBus. The numbers, as compared to the AnswerBus system are significantly better and have been achieved through simpler but effective techniques.

The precision numbers from both systems may seem to be low but they are not surprising since the precision, *in terms of relevance not presence of all keywords of the query,* of search engines as also cited in Kimberly *et. al.* (2002) is typically not more than 50%. The numbers are good also because the best performing question answering system on the TREC 9 dataset for which the TREC 9 questions are designed could answer only 65% of the questions while our system works on web. It is commendable since it is not necessary that all the answers are there on and also because the documents are not hand-picked for being structured and correct unlike TREC 9 document set.

Our numbers can be improved if we scan more documents rather than just the first 10 but that will reduce response times. Space doesn't permit to present elaborate experimentation results for individual queries but we present an example TREC query with the top five answers generated by the system.

Example

Where is Belize located?

The answers were:

1. Aqua Dives Belize is located just steps from your room.
2. Formally known as British Honduras, Belize is located below Mexico, east of Guatemala, and north of Honduras.
3. BELMOPAN The capitol of Belize is located in land 50 miles southwest of Belize City.
4. Belize is located on the Yucatan Peninsula in northern Central America.
5. Belize is located within the Tropic of Cancer in which most of the worlds greatest reefs can be found.

Although, all the above answers are quite good we do not consider answer 1 and 3 as *precise.* The reason being they do not specifically give information about *Belize*.

System	Answer Templates	Phrases	Keywords	Overall Precision
Our System	59%	57%	52.2%	55.82%
AnswerBus	-	-	-	34.88%

Table 1: Performance Comparison in terms of Precision

7 Conclusion and Future Work

The system presented in the paper adapts and reframes Question Answering techniques to scale them for the web. The techniques concentrate on dealing with various problems that web as a knowledge base poses and are simple but effective. The preliminary results from comparison with the existing Question Answering system show that considerable benefits are obtained in terms of precision through our system. We intend to extend the coverage of the system to all possible question types which require implementing answer template generation for all remaining question types. We also aim at putting this system on the web in near future.

References

1. Adrian Akmajian and Frank Heny *An Introduction to the Principles of Transformational Syntax*, MIT Press, Cambridge, Massachusetts, 1975.
2. C. Kwok, O. Etzioni, D. Weld. *Scaling Question Answering to the Web.* In "Proceedings of WWW10", Hong Kong, 2001.
3. E. Voorhees *Overview of the TREC-9 Question Answering Track,* in Proceedings of the ninth Text REtrieval Conference (TREC 9) p. 71, 2000.
4. Drs. Kimberly A. Killmer and Nicole B. Koppel *So Much Information, So Little Time. Evaluating Web Resources with Search Engines,* T.H.E. Journal, September 2002.
   http://www.thejournal.com/magazine/vault/A4101A.cfm
5. Fellbaum, Ch. (ed). *WordNet: An Electronic Lexical Database*. Cambridge: MIT Press, 1998.
6. M. Sanderson. *Word Sense Disambiguation and Information Retrieval,* In Proceedings of ACM SIGIR Conference, 17, pp. 142-151, 1994.
7. Oliver Mason. *QTAG-A portable probabilistic tagger*. Corpus Research, the University of Birmingham, U.K, 1997.
8. S. Brin and L. Page. *The anatomy of a large-scale hypertextual (Web) search engine*. In proceedings of the 7th International World Wide Web Conference (WWW7)/Computer Networks, 30(1-7):107—117, 1998.
   http://www.google.com

9. U. Hermjakob, E. H. Hovy, and Chin-Yew Lin. *Knowledge-Based Question Answering*. In "Proceedings of the 6th World Multiconference on Systems, Cybernetics and Informatics (SCI-2002)", Orlando, FL, U.S.A., July 14-18, 2002.
10. W.A. Woods. *Lunar Rocks in Natural English: Explorations in Natural Language Question Answering*. In A. Zampoli, editor, Linguistic Structures Processing, pp. 521-569. Elsevier North-Holland, New York, 1977.
11. Zhiping Zheng *AnswerBus Question Answering System*. In Proceeding of HLT Human Language Technology Conference (HLT 2002). San Diego, CA. March pp. 24 – 27, 2002.

    http://www.answerbus.com
12. Jay Budzik and Kristian J. Hammond. Learning for Question Answering and Text Classification: Integrating Knowledge-Based and Statistical Techniques. *AAAI Workshop on Text Classification.* Menlo Park, CA, 1998.
13. Boris Katz, From Sentence Processing to Information Access on the World Wide Web. *AAAI Spring Symposium on Natural Language Processing for the World Wide Web.* Stanford, California. 1997.
14. Rohini Srihari and Wei Li. Information Extraction Supported Question Answering. *Eighth Text REtrieval Conference (TREC-8).* Gaithersburg, MD. November 17-19, 1999.
15. Sanda Harabagiu, Marius Pasca, and Steven Maiorano. *Experiments with open-domain textual question answering*. *COLING-2000. Association for Computational Linguistics/Morgan Kaufmann*, Aug 2000.
16. E. Brill, S. Dumais and M. Banko. *An analysis of the AskMSR question-answering system.* In Proceedings of 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002), 2002.
17. Green, B., Wolf, A., Chomsky, C., and Laugherty, K. *BASEBALL: an automatic question answerer.* In Proceedings of the Western Joint Computer Conference, pages 219--224. Reprinted in Grosz et al. (eds), Readings in Natural Language Processing, 1961.
18. Steven Abney, Michael Collins and Amit Singhal. *Answer Extraction.* ANLP 2000.
19. David Pinto , Michael Branstein , Ryan Coleman, W. Bruce Croft , Matthew King , Wei Li , Xing Wei *Summarization and question answering: QuASM: a system for question answering using semi-structured data*. Proceeding of the second ACM/IEEE-CS joint conference on Digital libraries, July 2002.
20. Ask Jeeves

    http://www.askjeeves.com