A Simple Question Answering System

Richard J Cooper and Stefan M Rüger
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ, England
s.rueger@doc.ic.ac.uk

Abstract. We describe our simple question answering system written in perl that uses the CMU Link parser (Sleator and Temperley 1991), Princeton University's WordNet (Miller 1995), the REX system for XML parsing (Cameron 1998) and the Managing Gigabyte search engine (Witten, Moffat and Bell 1999). This work is based on an MSc project (Cooper 2000).

Introduction

The main task of question answering is providing a short answer to a natural-language query supported by a document in an underlying document collection. Many question-answering systems approach the problem from an information extraction angle, and ours is no exception. In the following, we will describe the structure and workings of our system. Section 1 is concerned with the off-line preparation of the documents for the pure information retrieval task of identifying potentially relevant paragraphs. Section 2 details the question-time processing.

1 Indexing

All documents from the document repository[1] are cleaned from SGML mark-up to obtain their raw contents. Dollar and pound signs are replaced by the words *dollars* and *pounds*, respectively (in line with the Financial Times archive which already uses these transformations). The raw news articles are then split into paragraphs according to the individual newspaper's indication. This could be an indentation after a newline (AP newswire, Foreign Broadcast Information Service and Wall Street Journal), a special marker (San Jose News), the use of a long line per paragraph (Los Angeles Times), or the fact that a line is not right justified that signifies the end of a paragraph (Financial Times).

The paragraphs are then fed into the Managing Gigabytes search engine as stand-alone documents together with the original reference number. In effect, we get a passage retrieval rather than a document retrieval.

2 Question processing

The actual question processing is executed as a long pipeline of perl modules which use XML, eg, to mark-up entities or to communicate other information between the modules. At the start of the pipeline is the question, eg, *Q206: How far away is the moon?* Figure 1 shows the structure of the data flow and the modules involved.

[1] News articles from TIPSTER and TREC CDs, see http://trec.nist.gov

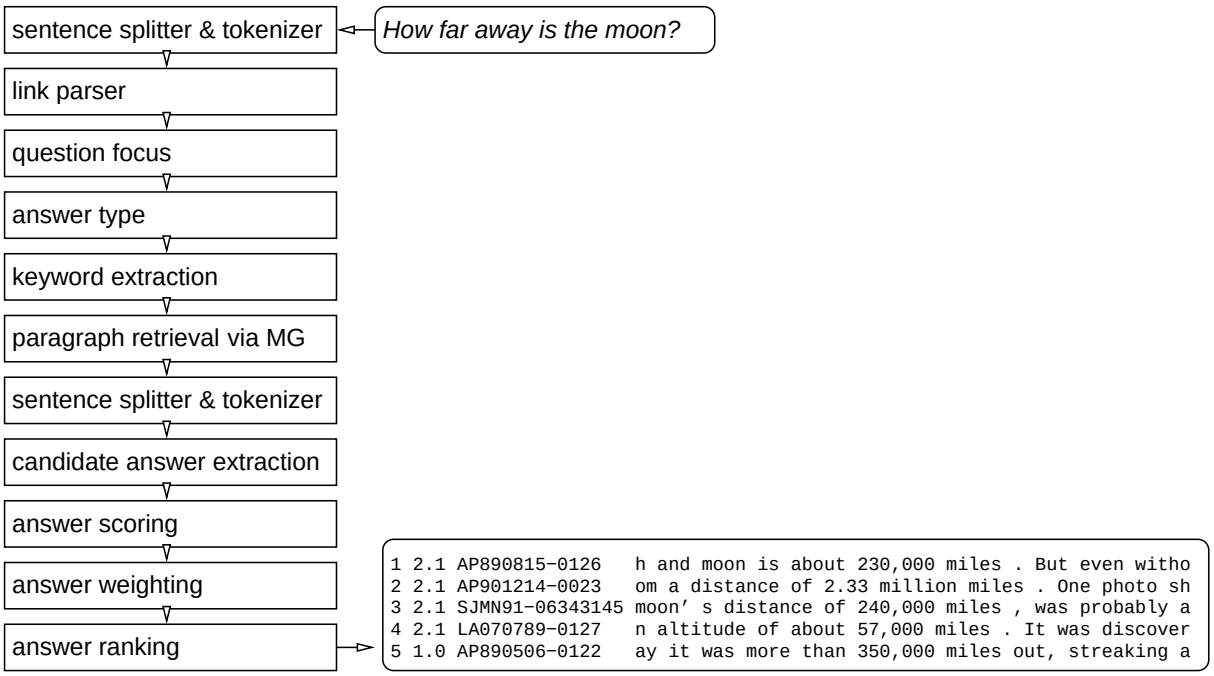

Figure 1: Question processing: data flow

2.1 Sentence splitter & tokenizer

The sentence splitter and tokenizer are actually implemented as two modules. The first marks up individual sentences using a set of heuristics for detecting the end of a sentence by a question mark, an exclamation mark, a full stop (if not preceded by a word in a list of abbreviations, such as Mr and Mrs), or the end of text. The tokenizer treats certain leading and trailing punctuation as separate entities, eg, "(", "[", ")", "]" or "?". Words containing digits are separated, if headed by a currency such as *DM* or *pounds* or trailed by an abbreviation such as *m*. Hence, *pounds20m* becomes the tree tokens *pounds*, *20* and *m*.

2.2 Link parser

The link parser is used to annotate the structure of the question. The link tree is appended to the tokenized question. For Q 206, eg, we get the following

```
<sentence><t n="1">How</t> <t n="2">far</t> <t n="3">away</t> <t n="4">is</t>
<t n="5">the</t> <t n="6">moon</t><t n="7">?</t><parse><pos n="2" pos="a"/><pos
n="4" pos="v"/><pos n="6" pos="n"/><link name="Xp" l="0" r="7"/><link name="Wq"
l="0" r="2"/><link name="PF" l="2" r="4"/><link name="MVp" l="2" r="3"/><link
name="SIs" l="4" r="6"/><link name="Ds" l="5" r="6"/><link name="RW" l="7"
r="8"/></parse></sentence>
```

2.3 Question focus

We identify the following question types which give a fairly clear indication of the type of answer: *when, who, where, whom, why, describe,* and *define.* For example, the answer to a *who* question is usually a person or a group of people. Other keywords or question words are less clear about the expected answer type: *what, which, how,* and *name.* For example, consider the following three *what* questions: *What **time** is the train arriving?*, *What **city** is the train stopping at?*, and *What is the name of the **driver** of the train?* This problem can be solved by defining a concept called question focus. The question focus is a phrase in the question that disambiguates it and emphasizes the type of answer being expected. For example in the three questions above the question foci are shown in bold. In the first two cases, the question focus tells us directly that a time and a city are being looked for. In the third case we know that a driver is a type of person and hence that a person's name is being sought. For the purpose of this system the question focus is defined as the first noun group that is not the word "name" if the question word is of an ambiguous type.

Normally question words start a question such as *when* in *When was Queen Elizabeth II born?* However sometimes they do not, eg, *In what city is the US Declaration of Independence located?* and *Macintosh Computers are made by whom?*, and we consider these cases as well.

For question 206, the mark-up `<questionFocus><t n="6">moon</t></questionFocus>` is inserted.

2.4 Answer type

Once the focus of the question has been found it is possible to decide what the answer type (or answer concept) should be. The question's question word determines how this module uses the question focus.

2.4.1 When, where, why, describe, define

When, where, why, describe and *define* are the easiest question words to process. They map directly into their answer type. The answer types for these question words are time, place, reason, description and definition, respectively.

2.4.2 Who, whom

In most cases *who* and *whom* question words imply an answer concept of person. However, there are two subtleties that are worth mentioning at this point.

Firstly, who questions could be looking for a group of people. This group could be a named group: *Who won the Premiership?* (Manchester Utd) or a list: *Who beat Fred in the 100m?* (Tom, Dick and Harry) or a combination: *Who beat England in the relay?* (America and Canada). At this stage the system does not correctly handle such questions, instead it always looks for a single person in response to a who question.

Secondly, there is an important exception to the rule that who questions always have an answer concept person. Consider, for example, *Who is Bill Gates?* This is an example of a who question that is looking for a description rather than a person. Accordingly, the rules for who and whom have the exception `(who|whom) (is|are|was) ProperNoun` which returns description as the answer type.

2.4.3 What, which, name

What, which, and *name* question words are extremely ambiguous when it comes to determining the question's answer concept. The answer type for these questions depends entirely on the question focus. The following pseudo-code illustrates the algorithm used:

```
until(wordNet contains questionFocus or questionFocus = "")
    Remove the first word of questionFocus
end until
if(questionFocus = "") then
    answerConcept = "name"
    exit
end if
hyponyms = the set of hyponyms of questionFocus from wordNet
if("person" is in hyponyms) then
    answerConcept = "person"
else
    answerConcept = questionFocus
end if
```

Following this idea further, one could also define the answer type as the list of direct hyponyms. Given the question *What type of bridge is The Golden Gate Bridge?*, the answer type *bridge* could be expanded (using WordNet) to include Bailey bridge, cantilever bridge, covered bridge, drawbridge, lift bridge, footbridge, overcrossing, pedestrian bridge, gangplank, gangboard, gangway, overpass, flyover, flypast, pontoon bridge, bateau bridge, floating bridge, rope bridge, steel arch bridge, suspension bridge, trestle bridge, truss bridge, or viaduct.

In our example of Q 206, `<answerType t="Length"/>` is added to the data stream.

2.4.4 How

With *how* questions the answer type is defined by what the word after *how* is — for example: *how old* (age), *how much* (quantity), *how long* (distance). If the word after *how* does not match any of the rules then the default answer type of *manner* is chosen, as in: *How did Socrates die?*

2.5 Keyword extraction

Using lists of place names, proper names and first names, entities are recognized and marked up with special symbols. This is simple non-structured knowledge that goes as far as identifying UK as a country, London as a city and Tony Blair as a person but the lists do not encode the relationship between these entities (and hence it is not known a priori that London is the capital of the UK). Recognized entities are among speed, temperature, money, place, city, country, person, year, time, length, reason, company, number, quoted and name, with the obvious meaning for the marked-up entity. In order to avoid inferences with potentially predefined SGML tags from the sources, we use special mark-ups of the form `<rjc99Person>` `Tony Blair` `</rjc99Person>`.

2.6 Paragraph retrieval via Managing Gigabytes

At the document paragraph lookup stage the information of answer type and possible keywords is used to extract documents (ie paragraphs) from the text database that might contain answers to the question.

The problem that the lookup stage has to overcome is the balance between getting sufficient documents to guarantee the presence of the answer and getting too many to process in a timely fashion. This problem defines the main task of the document paragraph lookup stage, which is to choose a "good" subset of the possible keywords with which to actually query the database.

This system defines "good" subsets by how many documents they retrieve. It attempts to find a subset that returns a number of documents within a given bound. As soon as such a subset is found it is chosen as the final query and the rest of the search is abandoned. If, after a complete search, no such subset is found, then the subset that is closest to that range is chosen. It attempts to reduce the amount of searching that need to be done by choosing the most likely subsets first, as defined by the weights assigned to each keyword. Once it has found a good subset, this module will query the document paragraph repository for real and add the texts retrieved to the output stream.

2.7 Candidate answer extraction

The paragraphs extracted from the document repository are sentence-split and tokenized in preparation for further processing. The Candidate Answer Extraction module's job is the mark-up of any regions of the texts that could be answers. The question's answer concept is looked up in WordNet and all of its hyponyms are found. A regular expression is then built by taking the disjunction of those hyponyms and any region of text that matches that regular expression is marked up as a candidate answer. There are a few exceptions to this rule which are detailed in the following sections.

2.7.1 Person

If the answer concept is person, the set of hyponyms would includes terms such as consumer, contestant, coward, creator, defender, guardian, and over 300 other words. This is not the intent as a person question is looking for a specific name of a person. So in the case of person questions, the hyponyms set is not computed and a regular expression that matches names is used instead.

2.7.2 Description

Describe, define and some who, whom and what questions result in an answer concept of description. Unlike answer types such and person and date, descriptions are very hard to define in terms of what words make them up. Any attempt to mark-up the descriptions in a piece of text would have to employ much more sophisticated NLP techniques than used in this system. Consequently, this system uses a much simpler technique to extract regions of text that could be descriptions. It has been noticed that when an entity is first introduced in a text it is often followed by a comma and then a description, as in *Bill Gates, Head of Microsoft said today ...* In light of this, descriptions are defined as everything between a comma and the next punctuation mark. Then when it comes to scoring the answers, descriptions that are immediately preceded by the thing that they are describing are scored highly. Even using such a simple and

naïve approach, good results are possible. For example, the top answers for the three questions below are shown in brackets: *Who is Steve Jobs?* (Co-founder and former chairman of Apple Computer), *Who is Steve Redgrave?* (Olympic gold medallist in 1984 and 1988), *Who is Nelson Mandela?* (President of the African National Congress).

2.7.3 General Cases

WordNet's coverage could never be great enough to cover every possible answer, especially with answer types that could contain an infinite number of answers, such as length. To deal with these eventualities, many of the common answer types have had extra subtypes added to WordNet which describe a regular expression for a general case. So, for example, *company* will match any of the explicitly named companies or the general case of anything that is a sequence of Proper Nouns ending in `(Ltd|Plc|Co|and Son|...)` and length will match any number followed by a unit of length such as *miles*, *km*, *ft*, etc.

2.8 Answer scoring

Once the candidate answers have been identified, a variety of heuristics are used to evaluate how likely that candidate is to be the real answer.

The following heuristics are used: (i) *score_comma_3_word.* If a comma follows the candidate answer then this score is the number of the three words following the comma that appear in the question; (ii) *score_punctuation.* Scores one if a punctuation mark immediately follows the candidate and zero otherwise; (iii) *score_same_sentence.* Computes the number of question words that are in the same sentence as the candidate answer; (iv) *score_description_before.* If the answer concept being looked for is a description then this score is the number of words immediately preceding the candidate answer that appear in the question; (v) *score_description_in.* Similar to *score_question_before* but counting question words that appear in the candidate answer.

Each heuristic is given a unique identifier, and at the end of this process each candidate will have associated with it a set of (id, score) pairs. Like most other things in this system, the scoring heuristics are implemented as pipeline modules. Each heuristic is a different module which scans its input for `<ca>` tags (for candidate answer), computes the appropriate score and adds a `<score>` tag. Each scoring heuristic is completely independent from the rest because, at this stage of the processing, the scores are being kept separate without regard for a single final score. This means that the order in which they activate is unimportant and that they can be removed or new ones added without affecting the others.

2.9 Answer weighting

Once all of the scores have been calculated they need to be combined into one final score. In this system the final score is simply a linear combination of all of the heuristic scores where the coefficients have been set by hand to reflect the perceived importance of the various scores: weight(*score_comma_3_word*) = 1.2, weight(*score_punctuation*) = 1.1, weight(*score_same_sentence*) = 1.0, weight(*score_description_before*) = 2.0, weight(*score_description_in*) = 1.0.

2.10 Answer ranking

Once every candidate answer has a weight they (and some of the characters around them which are to be used as context in the final answers) are extracted from the text, sorted by weight, assigned a rank and placed as final answers outside the document text.

At this stage one more thing happens. If the set of answers contains any duplicates then only the top ranking one is kept and all the other duplicates are removed. There are two possible modifications that could be made to this process that were considered, but not included in this version of the system. Firstly, only answers that are identical to higher ranked ones are removed. Ideally this would be replaced by a more powerful system which removed any answers that referred to the same entity as the first answer. A strong version of this system would probably require anaphora resolution of the document collection before they were split into paragraphs, and hence is beyond the scope of this system.

A weaker version, which removed any answers that probably referred to the same entity as the first answer, could be implemented using simple rules of abbreviation and word substitution. For example, Ms. L. Smith could be the same person as Miss Linda Smith, who could also be just Linda.

Alternatively, as some of the TREC-8 systems suggested, the presence of multiple instances of the same answer could be used to strengthen the likelihood of that answer being correct. We did not implement this.

3 Conclusions

Despite the apparent simplicity of our system, it compared favorably against other systems competing in TREC-9. Little use was made of natural language processing. The link parser analysis was only used for the proportion of questions that deal with ambiguous answer types; it was not used for the candidate answer extraction. We did not tune our system much, given the few training cases from the TREC-8 QA track. Although we did not (yet) carry out a failure analysis with the TREC-9 questions, we have reason to assume that there is scope for improvement in changing parameters, introducing a better ranking mechanism, or in deploying natural-language processing techniques.

References

Cameron, R. D. (1998). REX: XML shallow parsing with regular expressions. Technical Report 1998-17, School of Computing, Simon Fraser University.

Cooper, R. J. (2000). *High-Precision Information Retrieval.* MSc Thesis, Imperial College.

Miller, G. (1995). WordNet: A lexical database for English. *Communications of the ACM 38*(11), 39–41.

Sleator, D. and D. Temperley (1991). Parsing English with a link grammar. Technical Report CMU-CS-91-196, Computer Science, Canegie Mellon University.

Witten, I. H., A. Moffat and T. C. Bell (1999). *Managing Gigabytes.* Morgan Kaufmann Publishers.

Acknowledgements: This work was partially supported by the EPSRC, UK.