Ask Me Tomorrow: The NRC and University of Ottawa Question Answering System.

Joel Martin
National Research Council
Ottawa, Canada
joel.martin @nrc.ca

Chris Lankester
University of Ottawa
Ottawa, Canada
clank@site.uottawa.ca

Funny how one extra Perl sort function can make a score of 0.52[1] on the question answering task look a lot like 0. This paper describes our brute force approach to the question answering task and how we *did* achieve some success, despite formatting problems with our answer file. This paper also describes how we conducted automatic evaluations using the NIST judgment file on newly proposed answers.

A good question says what it is about and constrains the form of the answer. In other words, it specifies the *distinguishing context* for an answer and partially describes the kind of information that would count as an answer. The question "Who wrote the Declaration of Independence?" specifies that we are talking about someone who wrote a particular document, instead of engaging in some other action with respect to some other document. In addition, the question describes that we are looking for a "who" which must be a person, agency, or institution.

The process of finding an existing answer to a question means finding part of a document that matches the distinguishing context of the question and then extracting an answer of the right type from nearby. Ideally, the extracted answer would have the right sort of relationship to the context of the question.

Since this is our first year at TREC, and we were starting with no existing code, we decided to take a brute force approach to the problem. We divided the task into three phases. Phase I does a very high recall retrieval using the words in the question with the goal of discarding 90% of the document collection. Even with only a tenth of the documents, who wants to read them all? Phase II does an exhaustive scan of all 200-500 byte windows in the retrieved tenth, looking for strings with high similarity to the original question. This phase also ranks these text windows and adds extra points if an obvious answer type is nearby. Finally, Phase III was intended (and did a poor job of it) to extract the best answer of the right answer type from the best outputs from Phase II.

In the rest of this notebook paper, we first describe the three phases in more detail and then describe how we evaluated the system. We end with a discussion of the results and ideas for future work.

1.0 The Question Answering System (QA)

The question answering system was written in Perl and run on several Unix-based (Linux and Solaris) machines. Paragraph indexing and retrieval was

1 This is the score of our system based on the judgement of (non–NIST) human assessors .

done using an unmodified copy of MG 1.3 (Managing Gigabytes, Witten, Moffat, Bell, 1999). As noted before, the overall system was divided into three largely independent phases, each with a different goal.

1.1 Phase I: High Recall

Even though our system searches for answers in a brute force manner, it cannot exhaustively search through all the documents in the collections. Thus, we want to first select from all the documents those that are likely to contain all the answers. Achieving high recall is often quite difficult because it is hard to keep the list of retrieved documents small. In our case, we do not need the list of retrieved documents to be small yet. That is the task of the later phases. We only care that about 90% of the documents are excluded from the Phase I results.

Figure 1 shows the major substeps involved in performing Phase I. This is a description of the version of Phase I that was used for the answers submitted to NIST. After seeing the NIST test questions, we omitted the use of Wordnet to get synonyms as it was hurting the performance of Phase I.

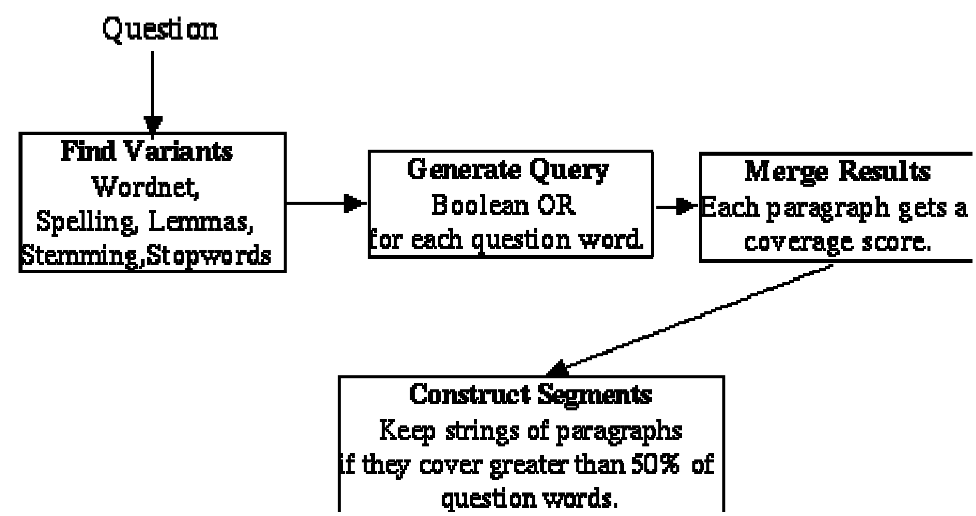

Figure 1: The steps involved in Phase I

In Phase I, each word of the question is first turned into a very long boolean query by disjoining many different forms of the word. For example, the word ‘question‘ would be turned into a query like: ”question OR query OR questioning OR questionable OR questoin”. Each of these queries is then sent to a search engine (Witten, Moffat, Bell, 1999). This results in one set of documents retrieved for each cloud of related words. In our system, each document is a single paragraph.

These sets of retrieved documents are then merged so that each document has a score reflecting how many of the non-stopwords in the question appear in that document. For instance, if the question were ”How are questions turned into queries?”, the paragraph immediately above this one would contain the words (or variants thereof): ‘questions‘ and ‘queries‘ and have a score of 2.

This score is calculated as an estimate of which documents contain more of the distinguishing context of a question. Finally, those paragraphs which are highly ranked are connected to adjacent high-scoring paragraphs to form segments. Segments are formed because the distinguishing context and the answer to the question may not appear in the same paragraph. Segments and paragraphs are discarded if they contain fewer than half of the question words.

1.2 Phase II: Higher Precision

The QA system assumes that the distinguishing context of a question will appear in a fairly small chunk of text and that the answer will be in close proximity. Thus, after locating those documents that contain the important words, the system should search for strings of words that densely pack those important words. If the words appear close together and possibly in the same order in a given paragraph, then that paragraph is better than another one where the important words are spread throughout the text.

Figure 2 shows the main steps of Phase II. First, Phase II augments the question with known answers stored in the thesaurus and with a tag indicating what type of question it is. Second, it builds a lookup table that stores a similarity score for a word to a word that appears in the question. This lookup table results in a less computationally expensive calculation of the similarities of many text segments to a given question.

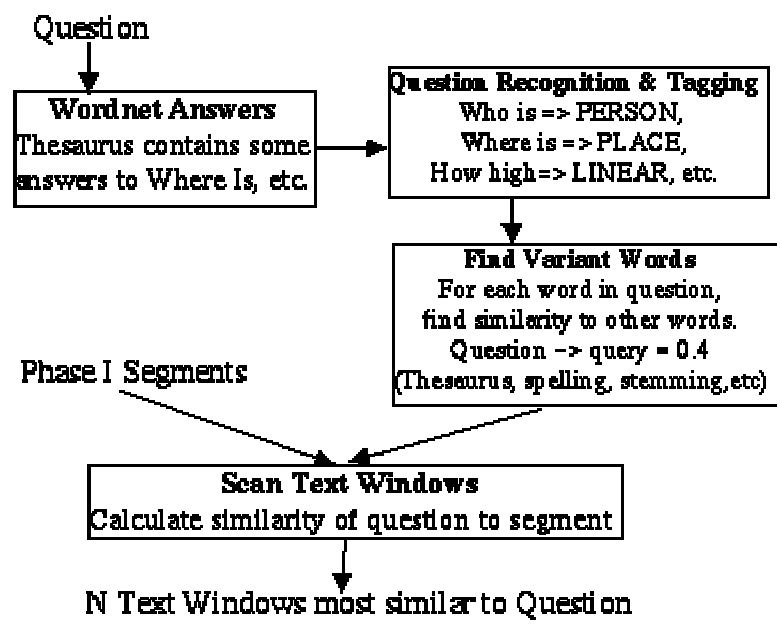

Figure 2: Steps in Phase II

Phase II assumes that there is a definition of similarity between two strings of words that is based on the similarities of the words, their relative position, and the addition or deletion of words. As well, there are several parameters that define the relative impact of each factor on an overall similarity score. This is much like an inverse sort of edit-distance.

Ideally, a large amount of training data would allow setting similarity parameters automatically. However, because only a small amount of training data was available for this competition, the parameters were set by hand.

1.3 Phase III: Answer Extraction

The results of Phase II contain many small windows of text that are very likely to contain the distinguishing context of a question. The task for Phase III is to scan the text in the window or just outside of it, looking for something that looks like an answer to the question. There is no magic algorithm for this phase. There must be a large and growing dictionary of question types and patterns for recognizing answers. If this is true, the success of Phase III should be less a result of the algorithm and more a result of having the right dictionary.

Figure 3 shows the main steps of Phase III. The first two steps of this phase should be done during indexing (and will be next year). This will save processing time and ensure consistent simplification of the paragraphs.

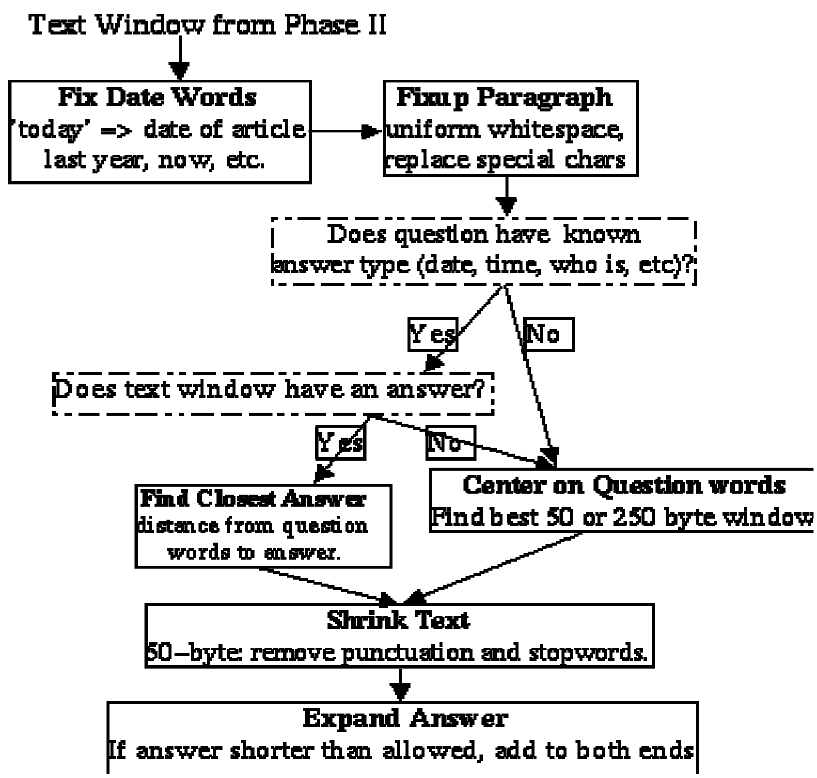

Figure 3: Steps followed in Phase III

Phase III assumes that there are question types and corresponding answer types. Furthermore, it assumes that the answers are in close proximity to the distinguishing context of the questions. Using a dictionary of question types, it checks whether the question is of a recognized type, such as ”Who is” or ”Who was”. If the question type is recognized, Phase III uses the corresponding pattern to find possible answers, such as ”John Smith”. When more than one answer is found, the answer that is closest to words that also appear in the question are better.

If, on the other hand, the question type is not recognized, Phase III finds the text window that covers as many words from the question as possible, trying to center on the distinguishing context. In either case, Phase III tries to maximize the amount of information contained in an answer for a given size of answer. If, for example, the answer must be 50 bytes long: a) the answer loses all its punctuation and stopwords, and then b) is expanded to be as close to 50 bytes as possible by adding tokens on the right and left of the window.

2.0 Using NIST Judgments for Automatic Evaluations

As we fix our QA system and want to test intermediate versions of it, we will not get human evaluations for every set of outputs. Instead, we need some automatic approximation to the official evaluations. This means that we must somehow compare the proposed answer strings from our new system with the NIST answers that appear in the judgment file.

Unfortunately, the NIST answers will not always match the lengths and boundaries of proposed answers. For example, two systems could choose the text window boundaries differently. One system might choose the answer substring: ”by Hugo Young (Farrar,”, whereas another might choose: ”Thatcher by Hugo Young”. Furthermore, several NIST answer strings have had some tokens removed, such as punctuation, SGML tags, or even some words. As a result, simple string searching will not give a good measure of the correctness of proposed new answers.

As a first pass, we can say that a proposed answer is correct if it fully contains a correct NIST answer. If the NIST answer is correct and a proposed answer contains all of it, the new answer must also contain the correct information. Of course, because of the problems of string matching, we will have to put the answers into a (closer to) canonical form. To do this, we replaced all strings of whitespace and punctuation with a single space. In addition, we replaced all special characters (&) and SGML tags with a single space.

Even with this notion of *contained-within,* there are at least three more problems. First, in the NIST answers, there are several examples where the same substring is deemed both a correct answer and an incorrect answer. For example, in question 54, one correct NIST answer is ”54 FBIS4-3997 1 22 April” and one incorrect NIST answer is ”54 FBIS4-19846 -1 22 April”. This may be because the judges looked at the substring in context and judged one document to contain the answer substring by accident. Second, in at least one case, a substring is correct if capitalized and incorrect if not capitalized: ”China” is right, ”china” is wrong. Third, although a correct answer may be contained in a proposed answer, it might be hidden among distracting information. For example, the proposed answer may list three proper noun phrases when only one is requested. In such a case, the proposed answer may be judged to be incorrect because the answer is buried.

To partially address these considerations, we augmented the definition of contained-within. Some apparent cases of contained-within can be *invalid* if the proposed answer also contains an incorrect answer. If this situation arises, we can use the source document to decide whether the proposed answer matches the correct answer or the incorrect one. More concretely (from question 103), suppose a proposed answer,P = ”Estonia last month, in which 900 people died. Norway is”, contains a correct NIST answer, C = ”900”. Further suppose that there is an incorrect NIST answer, I = ”900 people”, that is contained-within P and also contains (or is equal to) C. In this case, our only (automatic) recourse is to consult the source documents. Since the proposed answer came from the same document as the correct answer and not the same document as the incorrect answer, the proposed answer is rated as ‘correct‘.

```
flag <- contained-within(correct, proposed)
   correct_simp <- replace punctuation, spaces, SGML, &amp; with space
   proposed_simp<- replace punctuation, spaces, SGML, &amp; with space
   if (correct_simp is a substring of proposed_simp) then
         flag <- true
   else
         flag <- false
   end if

correct_flag <- isCorrect(P, correct_answers, incorrect_answers)
   correct_flag <- false
   foreach C in correct_answers (while correct_flag equals false)
         if (contained-within(C, P) then
                   if (not invalid(P, C, incorrect_answers)) then
                       correct_flag <- true;
                   end if
         end if
   end loop

invalid_flag <- invalid(proposed, correct, incorrect_answers)
   invalid_flag <- false
   foreach I in incorrect_answers (while invalid_flag equals false)
         if (contained-within(I, proposed) AND
                      contained-within(correct, I) AND
                      documentOf(I) equals documentOf(proposed)) then
                             invalid_flag <- true
         end if
   end loop
```

Table 1: The algorithms for automatic scoring of correct answers.

The specific rules applied are shown in Table 1. These rules are very conservative. They were derived to minimize the chance of rating a proposed answer correct when it is wrong (low false positive rate). It clearly misses many correct answers because there are no correct strings contained in the proposed answer. As in the above example (question 103), there are many correct substrings of ”a last month , in which 900 people died. Norway is”, but only a select few would be recognized as correct. If the judgment file continues to grow, this problem of many false negatives would gradually improve.

We must remember that the automatic score is generally only a lower bound on the performance of the question and answer system. If System A gets a score of 0.72 and System B gets a score of 0.56, that doesn‘t necessarily mean that System A is best. It could be that System A did a better job of matching the text windowing used to generate the NIST answers. As a side effect of how the NIST answers were generated, any official QA submission that was scored would receive very close to the same score by this automatic method as was reported by TREC (a very tight lower bound). Hence, we cannot easily use the results of the automatic evaluation to compare a new system against an official TREC submission.

Even though this metric is only a lower bound on question answering performance, it can still help improve our QA system. The automatic nature of this evaluation would allow it to be run many times in order to, for example, optimize the settings of several parameters. Suppose a system received a score of 0.72 by human evaluation and 0.63 by the automatic evaluation. If you can improve your system so that the automatic evaluation gives a score higher than 0.72, you know that you have a true improvement.

3.0 Results

Although our system's performance is difficult to compare to that of the other question answering systems in TREC, we can use the automatic evaluation to place a rough lower bound on performance. Two additional difficulties arise when doing this with the NIST answers. First, five of the NIST test questions have no answer in the judgments file and as a result, those questions cannot be marked correct. Second, because we are judging an answer correct if a NIST answer is contained-within it, it is very difficult to get an informative lower bound for the 50-byte task, (a 50-byte answer is not likely to contain many other answers).

We evaluated the original 250-byte submission using both the automatic evaluation method described in the last section and human judges. These scores are based on the full set of 198 questions, even though we are guaranteed to get 5 of them incorrect. Table 2 shows the results for the three phases.

	At least one answer in output	***NIST Rank Score***
Phase I	73%	---
Phase II	68%	---
Phase III	52%	0.4
Human	59%	0.52

Table 2: Results of the three phases for the answers submitted to NIST, August 1999

We have also re-run the first 100 questions after examining the test questions and answers and removing most of the sensitivity to synonyms. These results are shown in Table 3. All three phases improved markedly. In particular, Phase I now has 98 questions with at least one answer in the result list. Phase II is moderately better, as is Phase III. The (over) use of WordNet seemed to be a bad idea for the TREC test questions.

	At least one answer in output	*NIST Rank Score*
Phase I	98%	–––
Phase II	81%	–––
Phase III	67%	0.54

Table 3: Results of the three phases after removing WordNet.

Phase III is still a loose collection of heuristics and will require some organization to support further improvements. Many answers are lost because they appear one paragraph earlier than the identified answer or are lost because the date substitution is done incorrectly.

4.0 A Final Word

We are happy with our moderate success this year. Most of the product of our efforts is experience, and that will be applied in future competitions. As such, we hope that the QA track continues.

One way that the QA track might mature is to better define the types of questions that are included in TREC and those that are excluded. This year there was some process that selected appropriate questions from among submitted questions. Was this process random or where some questions omitted because their answers were not of the right form or they were difficult to find with typical search engines?

As noted in our discussion of Phase I, we originally designed the recall step to be robust to the use of synonyms in questions. Indeed, many of our ten submitted questions assumed some ability to recognize synonyms. Oddly enough, our system was quite good at handling this sort of question which did not appear in the TREC test questions.

Over the next few months, we expect to improve our question answering system. Besides optimizing the system to handle the TREC test questions, we will be generating a larger set of training questions. This should allow us to find more question-types and the corresponding answer-types.

We also want to speed up Phase I and Phase II. We will attempt to do this with as few minor changes to an existing search engine, as is possible. The goal will be to replace the function of Phase I and Phase II, while at the same time taking advantage of existing data structures in a search engine.

Phase III needs the most improvement. This phase attempts to extract an answer of the right type from paragraph-sized windows of text. Improvement in this phase may require a huge knowledge-engineering effort to find and characterize different types of queried information. This effort may require reproducing some of the effort that went into many of the systems designed for the MUC conferences.

5.0 Acknowledgments

The authors wish to thank Terry Copeck, Rob Holte, Ken Barker, Stan Matwin, and Stan Szpakowitz for discussions and assistance with questions,

answers, and the document collections.

This project is funded by an NSERC/NRC Research Partnerships grant: NSERC-NRC #653-022-96 ”Intelligent Information Access”

6.0 References

Witten, I. H., Moffat, A. & Bell, T. C. (1999). Managing Gigabytes: Compressing and Indexing Documents and Images. San Francisco: Morgan Kaufmann.