A Sys Called Qanda

Eric Breck, John Burger, Lisa Ferro,
David House, Marc Light, Inderjeet Mani

The MITRE Corporation

{ebreck, john, lferro, dhouse, light, imani}@mitre.org

Introduction

Our question answering system was built with a number of priorities in mind. First, we wanted to experiment with natural language processing (NLP) technologies such as shallow parsing, named entity tagging, and coreference chaining. We felt that the small number of terms in the questions coupled with the short length of the answers would make NLP technologies clearly beneficial, unlike previous experiments with NLP technologies on traditional IR tasks. At a more practical level, we were familiar with and interested in such technologies and thus their use would be relatively straightforward and enjoyable. Second, we wanted to use information retrieval (IR) techniques in hopes of achieving robustness and efficiency. It seemed obvious that many answers would appear in documents and passages laden with terms from the question. Finally, we wanted to experiment with different modules from different sites with differing input and output representation and implementational details. Thus, we needed a multi-process system with a flexible data format.

Driven by these priorities, we built Qanda,[1] a system that combines the finer-grained representations and inference abilities of NLP with IR's robustness and domain independence. In the following, we describe the Qanda system, discuss experimental results for the system, and finally discuss automating the scoring of question answering systems.

System Description

In broad strokes, Qanda processes a question as follows. It extracts the answer type from the question; e.g., PERSON is the answer type for *Who designed the Hancock Building in Boston?* At the same time, it hands the terms of the question to an IR search engine in order to retrieve relevant documents. Next it looks for elements in the retrieved documents that are of the right type; e.g., it might look for proper names of persons when answering a *who* question. Finally, it ranks these elements by comparing their contexts (i.e., sentences) with the question using term

---

[1] Qanda is pronounced *kwan•da*.

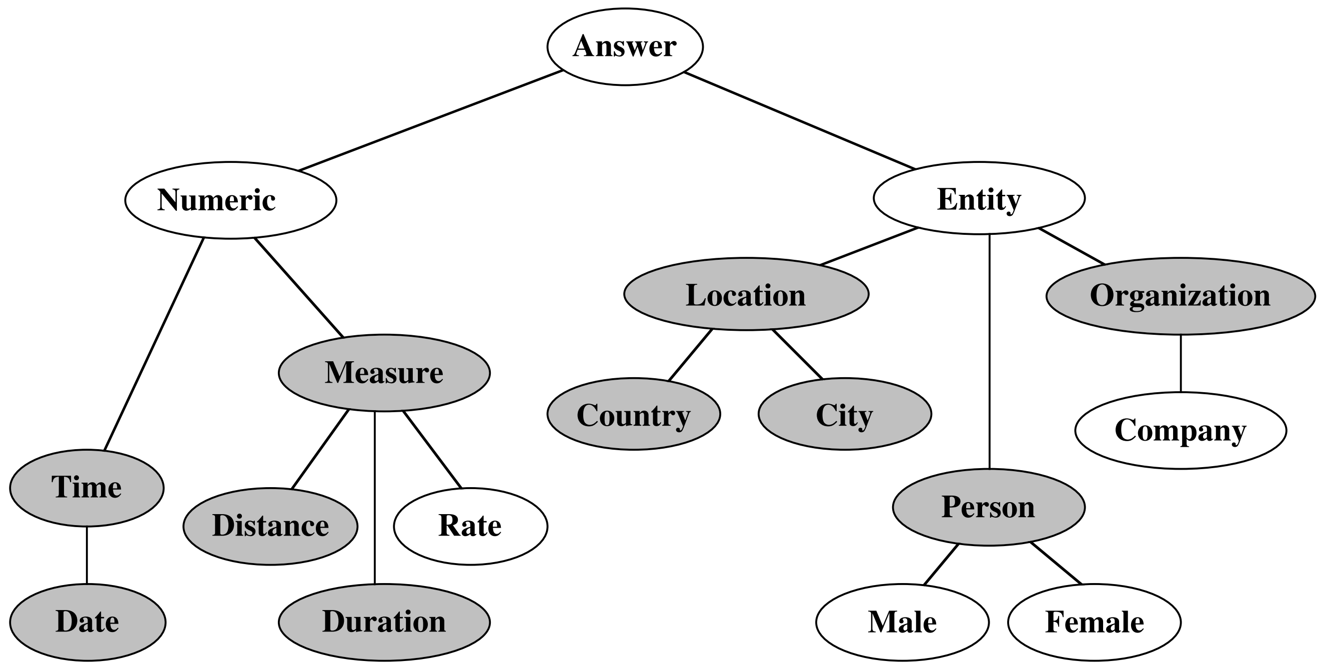

Figure 1: Answer type hierarchy

overlap. Thus, elements are returned which are of the right semantic type and whose contexts share some terms with the question. Based on all the references to an element and the corresponding contexts, an answer string is generated.

Let us look at each of these steps in more detail. The analysis of the question is performed using a set of specially designed patterns. These patterns are based on particular words and on part-of-speech tags. For example if the pattern **how (large|small|big)** is matched, the type MEASURE is returned. These answer types are arranged in the hierarchy shown in Figure1 (where the types actually used by the rest of the system are shaded). Our question analyzer also has the ability to return types constructed in part from words in the question (e.g., MANNER_OF(DIE) is the type returned for *How did Socrates die?*) but we are currently not making use of these more open ended types.

An important component of Qanda is the IR engine that finds candidate documents. We have used the MG system, and experimented with Smart, but for the results described in this paper, we used the documents provided from the AT&T system. A parameter of the overall system is the number of relevant documents to examine for answers—we used the top ten.[2]

---

[2] Our intent was to process the top 100 documents returned by the IR component. After submitting our results, we discovered a trivial bug that limited subsequent processing to the top ten.

The relevant documents are then searched for elements of the right type. A special purpose tagger was designed for each of the answer types that could result from question analysis. Thus, we had taggers for person proper names, measure phrases, dates, organization names, etc. Many of the types are recognized by the Alembic system (Vilain and Day, 1996).

We should note that for both the processing done by these taggers and for the analysis of the question, a number of preprocessing modules are used. They include a punctuation tokenizer, a sentence tagger, and a part-of-speech tagger (Aberdeen et al., 1995).

Next the set of elements of the desired answer type are ranked. First they are ordered by how well they match the answer type. Since these types are arranged in a hierarchy it is possible to consider elements to be of the right type if they are of types above the desired type in the hierarchy. For example, a candidate LOCATION matches even if an answer type of COUNTRY is desired. However, a COUNTRY is preferred, i.e., candidates with types at least as specific as the answer type are preferred to those with a more general type. The elements of the right type, which we will call answer hypotheses, are ordered further by considering the textual contexts in which they occur. More specifically, we score each sentence that an element occurs in by how many terms it shares with the question. We do not currently weight these terms in any way; however, they are stemmed with the Porter stemmer and a small set of stop words is used. Answer hypotheses are ordered further by preferring hypotheses that occur earlier in their documents.

Until now, we have remained vague about what an element (or answer hypothesis) is. Qanda makes use of coreference and thus an element is not an offset in a document nor is it a string. It is more abstract: it corresponds to an entity that is discussed in the document. Coreference makes the ranking discussed above more complicated in that multiple contexts need to be considered for each element, i.e., every sentence in which the element is mentioned. Qanda uses the score for an element's best context as the score for the element.

Once the answer hypotheses have been ranked, answer strings are constructed for them. Qanda does this by combining the longest realization of an element with its best-scoring context. Consider the question about the Hancock Building given above. Further, consider the answer hypothesis corresponding to the individual I.M. Pei, mentioned in the following distinct sentences:

> *I.M. Pei is a well-known architect.*
> *He designed the Hancock Building.*
> *Pei studied at MIT.*

For this answer hypothesis, Qanda constructs the following answer string: *I.M. Pei: He designed the Hancock Building.*

	250 byte	50 byte
Correct answer ranked 1	72	42
Correct answer within top 5	112	80
No correct answer	86	118
Mean RAR	0.434	0.281

Figure 2: Results on 198 TREC questions

If the above processing does not work, e.g., if no answer type is extracted from the question or no elements of the right type are found, Qanda uses the following fallback strategy: It looks for sentences in the relevant documents (from the IR search engine) that have a large term overlap with the question and returns these sentences as answer strings.

All of the modules mentioned above are glued together using a hub, which is a single executable that is configured to spawn a sub-process for each module and to set up a very simple file-based interprocess communication. XML is used for all data encoding. If the hub notices that a module is in an error state while processing a question it kills the module, restarts it, and goes on to the next question.

Finally, if for some reason no answer is found, as a last-ditch strategy, Qanda answers with the string *You're such a nice judge :-).*

Discussion of Results

Our TREC results are given in Figure 2. Note that Qanda found the 250-byte answer and ranked it within its top five responses 56 percent of the time. It ranked a correct answer first 36 percent of the time. With respect to the 50-byte responses the percentages are 40 and 21, respectively.

Next we discuss a preliminary error analysis. The purpose of this was to guide the future development of Qanda. Errors were divided into the following categories, with overlap:

- **Question Analysis**: The question analyzer failed to recognize the question type and the actual question type was one of those in the hierarchy.
- **Preprocessing**: One of the above-mentioned preprocessors failed. For example, a sentence boundary was incorrectly marked.
- **Question-Answer Context Comparison**: The answer was found but not ranked highly enough. Additionally, a different method of comparing the answer context with the question would produce a higher rank.
- **Numeric Expressions (Date,Time,Measures)**: An expression involving a number or referring to an element involving a number was misclassified.
- **Additional Answer Type Required**: The answer was of a type that is not part of our answer type hierarchy.
- **Coreference Resolution**: The extraction of the answer would have been facilitated by better coreference resolution of elements in the document.

- **Extra-Sentential Structure**: The extraction of the answer would have been facilitated by making use of rhetorical structure, discourse structure, document layout, etc. For example, the question *Why are electric cars less efficient in the north-east than in California?* would have been answered by our system if we would have recognized the relation between the sentence *John Williams … points out that electric cars are less efficient in the northeast than they are in California* and the following sentence which reads *The cold in the north-east hurts our range*.
- **IR**: The answer was not contained in any of the top ten documents.
- **Bugs**: Errors that were caused by problems that were thought of and addressed, but incorrectly implemented.

Figure 3 shows these error categories and the number of questions for which Qanda could find no answer (again, there is some overlap, due to multiple errors). Note that over half of the errors can be attributed to the IR engine not ranking an answer document highly enough, in this case, in the top ten. As mentioned above, our intent was to process the top 100 documents, which would have almost certainly reduced the number of such errors. Also note that numeric expressions are surprisingly important in this test set (63 of the questions had numeric answers), and that we had a number of problems with such questions. Finally, our comparison of questions with potential answer contexts must clearly be made more sophisticated.

Question Analysis	2
Preprocessing	10
Question-Answer Context Comparison	18
Additional Answer Type Required	2
Numeric Expressions (Date,Time,Measures)	17
Coreference Resolution	9
Extra-Sentential Structure	3
IR	47
Bugs	7
Other	4
All errors	86

Figure 3: Error categorization

Preliminary results on automating the evaluation of question answering systems

Automated evaluations are crucial for tight system development loops, which in turn often result in greatly improved system performance. We explored word-based precision and recall as a scoring metric for question answering systems. More precisely, we looked at scoring a system response to a question concentrating on the number of terms both in the response and in a human-prepared answer key for the question. The precision is the number of these overlap terms divided by the number of terms in the system response and the recall is the overlap divided by the number of terms in the answer key answer. The terms are stemmed and stop words are ignored. The answer key may contain multiple phrasings of a single answer and also multiple answers. The alternative in the answer key that provides the best F measure[3] for a system response is used to generate the precision and recall numbers for that response.

We are interested in how well these metrics correlate to a judge's binary scoring of responses. We have scored all of the participating systems' responses against an answer key, constructed by our chief annotator[4], based on her own knowledge, the responses, and the TREC corpus. Figures 4 and 5 plot a number of recall intervals on the horizontal axis and the number of systems responses that fall into the intervals on the vertical axis. The first graph is for responses deemed correct by the judges and the second for those responses deemed incorrect. We ignore precision since the system response length is roughly uniform. The results seem promising in that low recall correlates with incorrectness and high recall with correctness, with a coefficient of 0.84. However, we need to do a more careful statistical analysis and we need to explore the causes of the false positives and false negatives. A related study of answer key precision and recall metrics for automatic scoring of reading comprehension exams is described in (Hirschman, et al. 1999).

Conclusion

The mixture of NLP and IR that the Qanda system embodies has produced reasonable performance. Our error analysis indicates that performance can be increased by improving the modules that deal with numeric expressions and by improving the initial set of relevant documents considered. Simply considering more relevant documents for each question is likely to improve performance. In order to explore the impact of such parameter settings, we are using the automated evaluation method described above. We are also using this evaluation method to test configurations with and without certain modules (e.g., the coreference module). This will allow us to quantify the effects of that module.

---

[3] F measure is the harmonic mean of precision and recall.

[4] This individual has substantial experience in constructing language testing materials for adults but was not involved in the design or implementation of Qanda.

Acknowledgements

We would like to thank John Henderson, Warren Greiff, and Barry Schiffman for their help with the error analysis. We would also like to thank Lynette Hirschman for many helpful discussions.

References

Aberdeen, John, John Burger, David Day, Lynette Hirschman, Patricia Robinson and Marc Vilain 1995. MITRE: Description of the Alembic System Used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference*. Morgan Kaufman.

Hirschman, Lynette, Marc Light, Eric Breck, John D. Burger 1999. Deep Read: A Reading Comprehension System. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 325–332. College Park, Maryland.

Vilain, Marc and David Day 1996. Finite-State Parsing by Rule Sequences. *International Conference on Computational Linguistics (COLING-96).* Copenhagen, Denmark. The International Committee on Computational Linguistics.

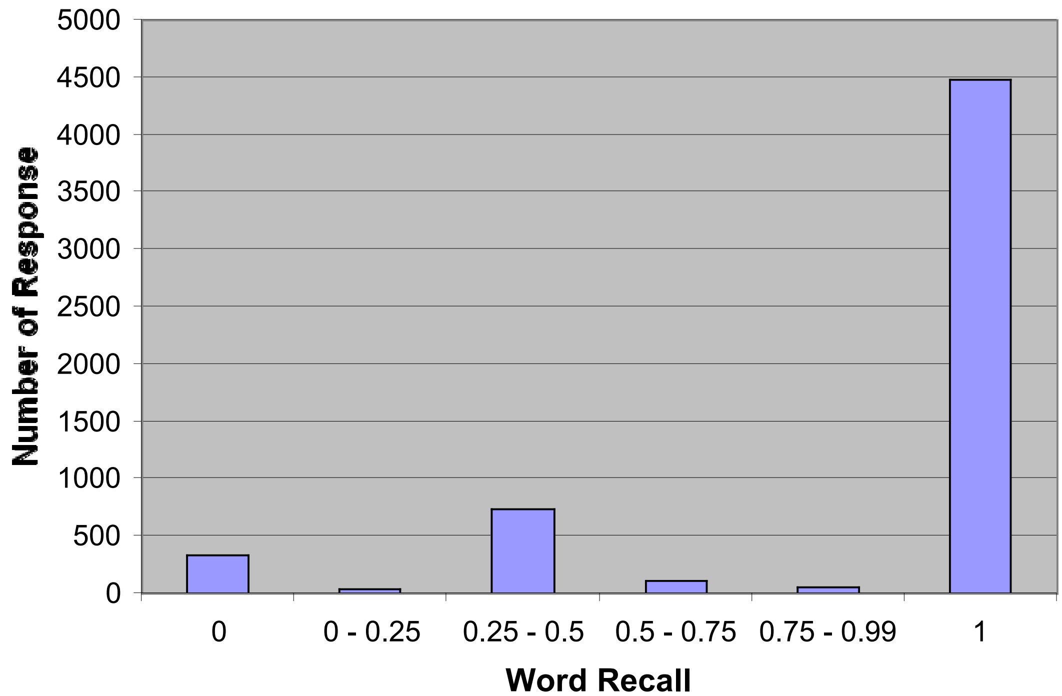

Figure 4: Automatic scoring for answers judged correct

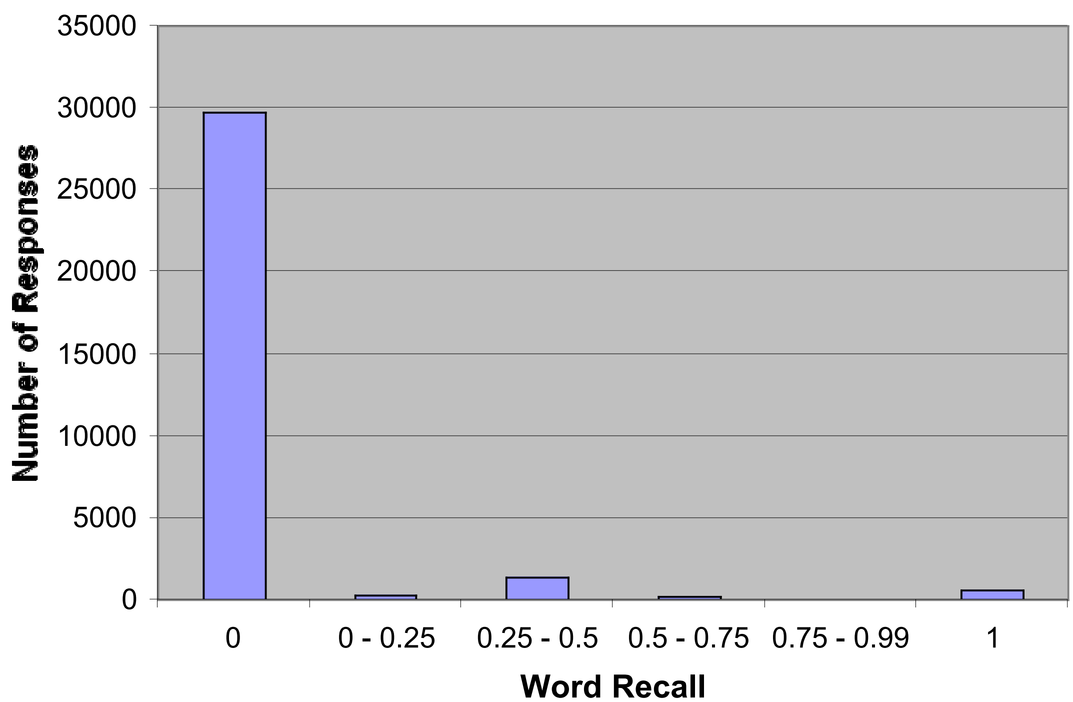

Figure 5: Automatic scoring for answers judged incorrect