Hybridization in Question Answering Systems

Jennifer Chu-Carroll, David Ferrucci, John Prager, Christopher Welty

IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
jencc,ferrucci,jprager,welty@us.ibm.com

Abstract

Question answering systems can benefit from the incorporation of a broad range of technologies, including natural language processing, machine learning, information retrieval, knowledge representation, and automated reasoning. We have designed an architecture that identifies the essential roles of components in a question answering system. This architecture greatly facilitates experimentation by enabling comparisons between different choices for filling the component roles, and also provides a framework for exploring hybridization of techniques – that is, combining different approaches to question answering. We present results from an initial experiment that illustrate substantial performance improvement by combining statistical and linguistic approaches to question answering. We also present preliminary and encouraging results involving the incorporation of a large knowledge base.

Introduction

Question answering systems can benefit from the incorporation of a broad range of technologies, including natural language processing, machine learning, information retrieval, knowledge representation, and automated reasoning. In order to experiment with the various technologies and measure their impact on an end-to-end question answering system, we have developed a flexible and easily extensible architecture that enables plug-and-play of system components to enable rapid prototyping of different system configurations. In addition, our new architecture supports a multi-agent approach to question answering by allowing multiple strategies for answering a question to be pursued in parallel and their results combined. This approach is motivated by the success of ensemble methods in machine learning, which has recently been applied to question answering, and has been shown to achieve significant performance improvement over their component systems [Chu-Carroll et al. 2002, Burger et al. 2002].

We begin this paper with a discussion of our question answering architecture, on which our PIQUANT question answering system is based, focusing on components that deviate from traditional approaches. We then discuss how PIQUANT instantiates these components to address question answering using multiple strategies and multiple resources. We present results from an experiment that shows significant performance improvement using this multi-strategy and multi-source approach. Finally, we present preliminary and encouraging results involving the incorporating of a large knowledge base (Cyc) into our question answering system.

Component Architecture Overview

The architecture adopted by our PIQUANT system, shown in Figure 1, defines several basic roles that components of a question answering system can play. The definition of each role includes a consistent interface that allows components implementing that role to be easily plugged into the system. This approach is not simply to facilitate good software engineering in a group, it allows *hybridization* at a fairly low development cost, and it also facilitates *experimentation* based on the choices available within the different component roles.

While our architecture can loosely be thought of as following the classic pipeline design of QA systems, which consists of question analysis, passage retrieval, and answer selection (see e.g., [Harabagiu et al. 2001, Prager et al. 2000, Hovy et al. 2000, Clarke et al. 2001]), we have introduced a number of components unique to our system that enable this experimentation and hybridization. These components can basically be divided into **processing components**, centered around answering agents, and consisting additionally of the question analysis and answer resolution components, and **knowledge source interfaces**, with provide access to both structured and unstructured information. Below we discuss the basic components of our system, emphasizing their deviation from traditional approaches.

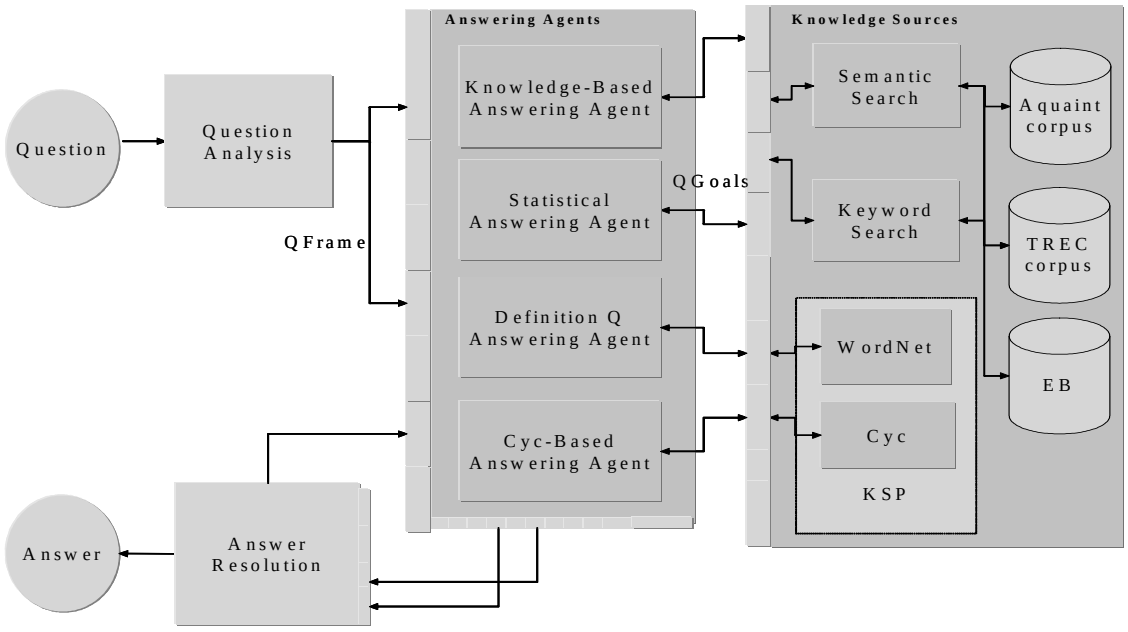

Figure 1 PIQUANT's System Architecture

Processing Components

Answering Agents

Answering agents are the central components in our architecture. They define strategies for finding answers to questions given information available from other components, such as question analysis, knowledge sources, and annotated corpora.

Answering agents get their input, the QFrame, from the question analysis component. Each agent in its independent operation is free to engage in a dialog with any number of knowledge sources through the knowledge source interfaces. The result from the knowledge source may be a direct answer to the question if the answering agent requested a database lookup of the answer. On the other hand, the answering agent may receive a set of relevant passages from a text corpus if a search query was issued. In the latter case, further processing by the answering agent's own **answer selection** component is needed to identify the answer to the question from the relevant passages. When complete, the answering agent delivers its results to the answer resolution component

Answering agents may differ at a fundamental level, e.g. one may use pure statistics to extract matching sequences from the corpus, and another may use a full parse of the question to extract semantic categories that appear in the annotated corpus. Another may use the predicate argument structure of the question to query a knowledge source.

Answering agents may agree on the fundamental level, but differ in terms of their applicability. For instance, some answering agents may be tailored toward addressing certain question types, such as definition questions, while others may be general purpose and can be invoked to answer all questions.

In addition to being able to define multiple answering agents that take different approaches to question answering, our architecture also allows multiple answering agents to be invoked in parallel. This approach allows us to experiment not only with different approaches to question answering strategies, but with *combining them* as well.

Question Analysis

The purpose of question analysis is to construct a QFrame that contains pertinent information from the question. Information in the QFrame is used to select one or more answering agents to address the question, and contains the necessary information for subsequent processing by the activated answering agents.

Information contained in the QFrame must be sufficient for all available answering agents to construct and issue queries to their respective knowledge sources. If the answering agent employs a search engine, then the query is a text search query, constructed by taking the original question, removing stop words, converting inflected terms to their canonical forms, adding synonyms or closely related words where appropriate, and building the correct syntactic form for the search engine query. If the answering agent uses a structured resource such as an

ontology, database or inference engine, the query may be in a logical form.

Answering agents that produce text passages will call answer selection to process them; in order to extract candidate answers, answer selection needs to know the semantic type of the entity being sought. Typical QA systems define anything from a dozen or so to one or two hundred semantic types for purposes of question classification [Ittycheriah et al. 2001, Hovy et al. 2001]. These can range from the broad MUC-type classes PERSON and PLACE, corresponding to "Who" and "Where" questions, to specific subtypes, such as PRESIDENT and CITY, or to non-proper-named entities, such as COLOR and ANIMAL. The expected semantic type of the answer is also determined by the question analysis component.

Answer Resolution

The task of the answer resolution component is to combine and reconcile results returned by multiple answering agents. Given the pipeline architecture traditionally employed by QA systems, resolution between two answering agents can be performed at multiple points in the pipeline. So far, we have implemented two answer resolution interfaces. First, two answering agents may be allowed to carry out their respective end-to-end processes and their individual final answers combined by answer resolution. Second, the search passages retrieved by one answering agent can be considered by the other answering agent as additional candidate passages in its own answer selection process [Chu-Carroll et al., 2002].

Knowledge Source Interfaces

The purpose of the knowledge source interfaces is to provide a uniform interface to all answering agents for knowledge access, regardless of the actual knowledge sources. We distinguish between access to unstructured knowledge, currently limited to text corpora, and access to structured knowledge such as databases and ontologies.

Access to Unstructured Knowledge

Traditional QA systems attempts to locate answers to questions in a large text corpus. In terms of knowledge access, this requires formulating a query from the question issuing the query to a search engine, and receiving a set of relevant documents/passages in return. In Figure 1, we showed two instantiations of search, the traditional **keyword search**, and **semantic search**, which allows search terms to be either keywords or semantic categories of terms. In order to enable search on semantic categories, the corpus must be indexed with such information in a process described below.

Semantic Annotation. Prior to building a text index, we analyze the entire corpus with a set of text analysis engine (TAE) components [Byrd and Ravin, 1999], and then generate a special purpose index of the semantic entities discovered by the engine. This semantic index can then be used by answering agents to search the corpus for classes of terms that may not match any particular string. For example, the question:

When (or In what year) did Beethoven die?

can be turned into a search for the terms, "Beethoven", "die" and the semantic category *YEAR*, which then matches passages containing an instance of a year (but note not necessarily the word "year"), "Beethoven" and "die". Subsequent processing in answer selection and answer resolution seek to identify those entities of semantic type YEAR that are actually about Beethoven dying. The two-pronged process of indexing semantic labels, and including them in queries for search, is called Predictive Annotation [Prager et al, 2000].

An obvious benefit of our component-based approach is that we can easily experiment with and compare different annotation techniques by keeping the rest of the components of the QA system fixed and changing only the TAEs producing the index. Thus we could, for example, measure the overall impact on QA performance of using statistical vs. rule-based annotators.

Structured Knowledge Adapters

We have identified many uses of *structured* knowledge in the QA process. Knowledge can be used at corpus annotation time to, e.g. perform simple inferences from class hierarchies or entail obvious consequences of described actions. Knowledge can be used during question analysis to guide a parse, identify question categories, etc. Knowledge can be used by answering agents as a direct source of answers to questions. Knowledge can be used during answer resolution to prune out ridiculous answers, and to help score candidate answers. Finally, knowledge can be used to classify answering agents themselves, based on, e.g., areas of expertise or question category.

Due to the many ways knowledge may be employed, we have defined a key component of the system responsible for adapting various structured knowledge sources into the QA architecture. This component, OntASK (Ontology-based Access to Structured Knowledge) insulates the other components of the QA system that may have need of structured knowledge from the multitude of data formats, access mechanisms, representation languages, reasoning services, and ontologies that consumers of existing structured knowledge sources must be acutely aware of.

OntASK presents a clean and consistent interface that supplies a single data format, representation, and ontology to the other components of the QA system. There are only two major interfaces that other QA components need to be

directly aware of: knowledge-source adapters (KSAs) and a KSA directory service (KDS).

A KSA is a single component that is responsible for adapting one or more knowledge sources into the format, language, and ontology that the QA components understand. In the PIQUANT project, for example, we have chosen a predicate logic form with a simple syntax based on KIF (KIF is, for our purposes, a serialization of first order logic that allows logical formulae to be specified within the ASCII character set), and defined an ontology which is derived directly from the semantic categories our predictive annotation engine recognizes. One of the simplest predicates our ontology specifies is "ISA", that is, the relation between two terms specifying that the first term is more specific than the second, e.g. ISA(HORSE, MAMMAL).

There are a number of existing knowledge sources that contain information in this form, for example WordNet [Miller, 1995] and Cyc [Lenat, 1995] In WordNet's ontology, however, this relation is called "hypernym" and in Cyc it is called "genls". In addition, WordNet uses a database format with a fixed number of tables, and Cyc uses a KR language with a LISP-like syntax called CycL. Rather than requiring each QA component that wishes to capitalize on the knowledge in Cyc and WordNet to implement access to these different sources, we simply provide a set of components that adapt their format, language, and ontology into that of the system. Thus, all QA components that need information about the general terms of horse, submit a query to OntASK such as "ISA(HORSE, ?x)", and the Cyc adapter returns information about the "genls" of the Cyc term "Horse", and the WordNet adapter returns information about the "hypernyms" of the word Horse.

The KDS is a web-services based component that permits the dynamic deployment of KSAs that adapt new knowledge sources or new pieces of knowledge sources. KSAs register themselves within the OntASK framework by listing the *predicates* they support. When a query is made, the KDS finds all the KSAs that support the predicate in the query, and those KSAs are dispatched. If no KSA supports the predicate desired, a special status code is returned.

This approach was taken because, generally speaking, in the PIQUANT QA system most of the queries to the structured knowledge sources are generated from the natural language questions. We cannot predict ahead of time every predicate that might be generated during the question analysis process, and we do not wish to require that all other QA components know *every* predicate the OntASK framework provides. From the perspective of the other QA components, the set of supported predicates in the ontology is open-ended, and may change over time. Thus, if we recognize a predicate that is being requested in queries but is not supported, we may take action and provide a KSA that supports it, and the other QA components do not need to change. Thus, this approach only requires that the QA components handle the special "unsupported predicate" status code.

Multiple Answering Agents

PIQUANT currently includes answering agents that adopt two different processing strategies, and consult three different text-based knowledge sources and a number of structured information sources through KSAs. The two processing strategies both employ the traditional pipeline architecture, but utilize fundamentally different approaches within the individual components for question answering.

In the rest of this section, we briefly describe the characteristics of the answering agents we employed, as well as our experimental setup and results.

Knowledge-Based Answering Agent (KBA)

Our first answering agent utilizes a primarily knowledge-driven approach to question answering, based on the predictive annotation mechanism described earlier. The basic architecture follows the traditional pipeline, with most components employing rule-based processing mechanisms [Prager et al., 2000; Prager et al., to appear].

Statistical Answering Agent (SA)

The second answering agent we employed in our multi-agent architecture is a statistical question answering system [Ittycheriah et al., 2001]. This agent again employs the pipeline architecture, and the individual components are statistically trained based on the maximum entropy framework.

Experimental Setup

To assess the effectiveness of our multi-agent architecture, we conducted experiments using questions from the TREC10 Question Answering track. We selected from the original 500 questions those questions 1) whose answers existed in the reference corpus (henceforth referred to as the TREC corpus), and 2) which was not a *definition* type question.[1] This results in a test set of 313 questions.

Our experiments are designed to evaluate the impact of answer resolution using both our answer resolution interfaces described earlier. First, we evaluate the impact of integrating results from multiple corpora using the

[1] Definition questions (e.g. "Who is Colin Powell?" and "What is mold?") are handled by a special process in the knowledge-based answering agent [Prager et al., 2001], which greatly reduces potential contribution from other answering agents. Thus, we exclude such questions in our current evaluation.

knowledge-based answering agent. More specifically, we configured the knowledge based answering agent to use the TREC corpus as its primary corpus from which answers will be given, and to use the new AQUAINT corpus as a knowledge source from which additional supporting evidence may be found. Next, we assess the contribution of merging the results of two different answering agents both at the passage level and at the answer level. Here we merged in the passages retrieved by the statistical answer agent from the TREC corpus as additional passages from which the knowledge based answering agent will ultimately select its answers. Additionally, the answers generated by the statistical answering agent are used to adjust the confidence of the final answers produced by the knowledge based agent.

Results

For each of the 313 questions in the test set, PIQUANT generated its top answer along with the system's confidence in it being a correct answer. These 313 answers are then rank ordered based on the confidence value.

Two evaluation metrics are used. First, we measure the number of correct answers out of the 313 answers given by PIQUANT. Second, we adopt the new TREC average precision metric in which a confidence weighted score is computed as follows:

$$AP = \frac{1}{N} \sum_{i=1}^{N} \frac{\#Correct\ upto\ i}{i}$$

where N is the total number of questions (in this case 313).

	# correct	avg prec
KBA (TREC)	124	0.595
SA	116	0.569
KBA (TREC+AQUAINT)	138	0.647
KBA (TREC+AQUAINT) + SA	151	0.714

Table 1 Experimental Results

Table 1 shows the results of our experiments. The first two rows of the table show that our baseline answering agents perform quite comparably in terms of both the number of questions correctly answers and in average precision. Furthermore, enabling the knowledge based answering agent to consult an additional corpus for supporting evidence resulted in fairly substantial gain using both metrics. Comparing the results in row 3 to the baseline in row 1 shows an 11.3% relative improvement using the "number correct" metric, and an 8.7% relative improvement using the average precision metric. In addition, performing answer resolution using two strategically-independent answering agent resulted in even further performance gain. Comparing the results in the last row to the baselines in rows 1 and 2 shows a 21.8% relative improvement using the "number correct" metric, and a 20.0% relative improvement using the average precision metric over the better performing baseline system.

NLP and Sanity Checking in Answer Resolution

One of the ways in which the knowledge source adapters have been integrated into our current system is in the so-called "sanity checking." Sanity checking is motivated by the observation that people often use their real world knowledge to rule out answers that appear quite improbable, such as the weight of a gray wolf being 300 tons. Sanity checking utilizes a structured knowledge source (in our current implementation, Cyc) to determine whether a top-ranked answer selected for a question is *sane* or *insane*. This process can be used to eliminate incorrect answers that may otherwise be difficult to rule out by non-knowledge-based components.

The answer selection component of our knowledge-based answering agent ranks candidate answers to passages based on three criteria: 1) semantic type of candidate answer, and 2) weighted grammatical relationships, and 3) frequency of answer. The first criterion checks for a match between the semantic type of the answer as inferred from the question and the candidate answer itself. The second criterion measures how closely the grammatical relationships in the question are mirrored by those in the passage surrounding a candidate answer. The third criterion captures the notion that identical answers occurring in different contexts reinforce one another [Clarke et al., 2001]. These three criteria combined generate a confidence score for each candidate answer and the agent passes on the top *n* candidate answers to the answer resolution component for confidence re-ranking.

The sanity checking process is invoked after the answer resolution component re-ranks all its candidate answers. Currently, the INRANGE operator is supported, which applies to most questions seeking numerical values, such as "*What is the population of the United States?*" and "*How high is Mount Everest?*" The sanity checker is invoked with the expected semantic type of the answer ("POPULATION" in the first example above), the focus of the question ("United States") and the system's proposed answer ("X people"). It currently returns one of the following verdicts: "in range", which results in an increase in the confidence associated with the answer, "out of range", which results in the answer being eliminated, or "don't know", which has no effect.[2]

Consider the TREC11 question, "What is the population of Maryland?" Prior to sanity checking, our system's top

[2] In future work, we will extend the sanity checker to return the probability of a candidate answer being correct, based on a normal distribution around Cyc's knowledge of the correct answer.

answer is "50,000", from the sentence "Maryland's population is 50,000 and growing rapidly." This would otherwise be an excellent answer if it were not for the fact that the article from which this passage is extracted discusses an exotic species called nutria.[3] By employing sanity checking, however, our system was able to consider that answer "out of range", and return an initially lower-ranked correct answer "5.1 million" instead with high confidence.

Conclusion

We have designed an architecture that identifies the essential roles of components in a question answering system. This architecture greatly facilitates experimentation by enabling comparisons between different choices for filling the component roles, and also provides a framework for exploring hybridization of techniques – that is, combining different approaches to question answering. We have implemented several components that make specific choices about how to do fill these roles. We presented results from an initial experiment combining statistical and linguistic approaches to question answering that showed performance improvement of over 20% according to two standard evaluation metrics. We also presented preliminary results involving the incorporation of a large knowledge-base, Cyc, in validating the answers generated by search.

Acknowledgments

We would like to thank Abe Ittycheriah and Salim Roukos for making their system and results available to us for experimental purposes, Stefano Bertolo for his help with Cyc integration, Ruchi Kalra for ontology population, and Krzysztof Czuba for helpful discussions, and Alan Marwick for his comments on an earlier draft of this paper. This work was supported in part by the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) Program under contract number MDA904-01-C-0988.

References

Byrd, R. and Ravin, Y. Identifying and Extracting Relations in Text. In Proceedings of NLDB 99, Klagenfurt, Austria, 1999.

Burger, J., Ferro, L., Greiff, W., Henderson, J., Light, M., and Mardis, S. MITRE's Qanda at TREC-11. In Proceedings of TREC-2002. Gaithersburg, MD. 2002.

Chu-Carroll, J., Prager, J., Welty, C., Czuba, K., and Ferrucci, D. A Multi-Strategy and Multi-Source Approach to Question Answering. In Proceedings of TREC-2002, Gaithersburg, MD, 2002.

Clarke, L.A., Cormack, G.V. and Lynam, T.R. Exploiting Redundancy in Question Answering. In Proceedings of SIGIR 2001, New Orleans, LA, 2001.

Harabagiu, S., Moldovan, D., Pasca, M., Mihalcea, R., Surdeanu, M., Bunescu, R., Girju, R., Rus, V., and Morarescu, P. The Role of Lexico-Semantic Feedback in Open-Domain Textual Question-Answering., In Proceedings of ACL 2001, Toulouse, France, 2001.

Hovy, E., Gerber, L., Hermjakob, U., Junk, M., and Lin C-Y. Question Answering in Webclopedia. In Proceedings of TREC-2000, Gaithersburg, MD, 2000.

Hovy E., Hermjakob U. and Lin C-Y. The Use of External Knowledge in Factoid QA. In Proceedings of TREC-2001, Gaithersburg, MD, 2001.

Ittycheriah, A., Franz, M., Zhu, W-J, and Ratnaparkhi, A. Question Answering Using Maximum Entropy Components. In Proceedings of NAACL. 2001.

Lenat, D.B. Cyc: A Large-Scale Investment in Knowledge Infrastructure. Communications of the ACM 38, no. 11, Nov. 1995.

Miller, G. WordNet: A Lexical Database for English. Communications of the ACM 38(11) pp. 39-41, 1995.

Pasca, M.A. and Harabagiu, S.M. High Performance Question/Answering. In Proceedings of SIGIR 2001, New Orleans, LA, 2001.

Prager, J., Brown, E., Coden, A., and Radev, D., Question-Answering by Predictive Annotation. In Proceedings of SIGIR, 2000.

Prager, J.M., Radev, D.R. and Czuba, K. Answering What-Is Questions by Virtual Annotation. In *Proceedings of Human Language Technologies Conference*, San Diego CA, March 2001.

Prager, J., Chu-Carroll, J., Brown, E., and Czuba, K. Question Answering Using Predictive Annotation. To appear in Advances in Question Answering, 2003.

Wacholder, N., Ravin, Y., and Choi, M. Disambiguation of proper names in text. In Proceedings of the Fifth Applied Natural Language Processing Conference, Washington, D.C., 1997.

[3] Alternatively, incorrect answers can be eliminated by performing more sophisticated processing in the answer selection component. For instance, "50,000" will be considered an incorrect answer in the population of Maryland example if global discourse context is taken into account. We plan to investigate this issue in future work.