MITRE's Qanda at TREC-14

John D. Burger and Sam Bayer
The MITRE Corporation
{john, sam}@mitre.org

Introduction

Qanda is MITRE's TREC-style question answering system. In recent years, we have been able to apply only a small effort to the TREC QA activity, approximately two person-months this year. (Accordingly, much of this discussion is strikingly similar to prior system descriptions.) We have made some general improvements in Qanda's processing, including genuine question parsing, generalizing answer selection to better handle variant question types like lists and definition questions, and better integration with a maximum entropy answer scorer, both in training and at run time. We have also attempted to better integrate the results of question processing and document retrieval.

1. TREC-14 system description

Underlying architecture

Qanda uses a general computational infrastructure for human language technology called the Annotation Management System (AMS). AMS is a flexible library for pairwise interaction between language processors, based on the Catalyst infrastructure used in previous versions of Qanda (Burger 2004, Burger & Mardis 2002, Nyberg et al. 2004). Where Catalyst was specifically designed for fast processing, AMS is designed for compatibility and reuse. Essentially, AMS provides an extensible wrapper between a consistent internal programming model for language processors and the wide range of ways the language processor can be invoked, as well as the wide range of possible annotation formats and storage types. Philosophically, it is similar to IBM's UIMA infrastructure (Ferrucci & Lally 2004), without the benefits and drawbacks associated with the strong programming assumptions that UIMA makes. In comparison with Catalyst, AMS allowed us to make rapid changes to our system configuration, and introduce new language processing components with relatively little effort.

Major system components

Qanda has a by now shop-worn QA architecture, which proceeds in several phases. Questions are analyzed for expected answer types, as well as keywords to use in forming an IR query. Documents are retrieved using an IR system and are then processed by various taggers to find entities of the expected types in contexts that match the question. For TREC-14, we enhanced the analysis phases for both questions and document passages, including detailed parsing of questions. Below we describe each of the major components in turn.

- *Common question and document processing*: This phase consists of several steps: tokenization, sentence boundary detection, part of speech tagging (Ratnaparkhi 1996), morphological analysis (Minnen et al. 2001), and tagging of named persons, locations and organizations (named entities), as well as temporal expressions, for which Qanda uses Phrag (Burger et al. 2002), an HMM-based tagger.
- *Question analysis*: After the common initial phase of analysis, questions are chunked and parsed, and salient features of the meaning of the question are extracted. See Section 2 below for more detail.
- *IR wrappers*: AMS components have been written for several IR engines, taking the results of the question analysis and formulating an IR query. We continue to use the Java-based Lucene engine (Apache 2002). Lucene's query language has a phrase operator, and also allows query components to be given explicit weights. Qanda uses both of these capabilities in constructing queries from the information extracted from the question. For TREC-14, the top 50 documents were retrieved**.**
- *Passage processing*: After the retrieved documents pass through the common analysis phase, Qanda assigns a preliminary score to each sentence by summing the log-IDF (inverse document frequency) of each word that occurs in both the candidate sentence and the question. Those sentences with a low score are not processed by most of the system. This step is performed to reduce the cost of more expensive downstream components.
- *Fixed repertoire taggers*: We have a simple facility for constructing AMS taggers from fixed

word- and phrase-lists. These were used to re-tag many named locations more specifically as cities, states/provinces, and countries. Qanda also identifies various other (nearly) closed classes such as precious metals, birthstones, several animal categories (e.g., state bird), and so on. Different taggers are applied to the question and to the retrieved passages.

- *Numeric tagging*: A fixed repertoire tagger is run on the retrieved passages to identify words and phrases denoting units of measure, and then a simple pattern-based tagger combines these with numeric expressions to identify full-fledged measure phrases, as well as currency, percentages and other numeric phrases.
- *Overlap*: The question is compared to each sentence, and a number of overlap features are computed, some in terms of various WordNet relations (see Section 3).
- *Answer collection and ranking*: Candidates are identified and merged, a number of features are collected, and a score is computed (Section 3).
- *Answer selection*: A final component down-selects the candidates and generates the actual answer strings. For factoid questions, this is simply the highest-scoring phrasal candidate, but definition and list questions require other processing, as detailed in Section 4.

As described above, all of these components communicate by consuming and producing stand-off annotations. A separate declarative facility is used to indicate which components are interested in consuming which annotations, and AMS arranges for the components to be connected.

2. Question analysis

In previous TREC evaluations, Qanda performed a limited analysis of the questions. We tagged for part-of-speech and named entities, and also applied a simple fixed-repertoire tagger that maps head words to answer types in Qanda's ontology, using a set of approximately 6000 words and phrases, some extracted heuristically from WordNet, some identified by hand. For TREC-14, we added a detailed parsing phase using MITRE's Carafe (Wellner, 2005) conditional random field chunker and the Pro3Gres dependency parser from the University of Zurich (Schneider et al. 2004), and performed a heuristic analysis on the resulting structure to extract various dimensions of the question.

Because gold-standard data for questions is scarce, many of our corpus-based tools require a repair phase to address some of the more egregious misinterpretations of questions as declarative statements. For instance, it is not uncommon for a part of speech tagger that has been trained on declarative data to attempt to tag questions like *Who does John love?* as if *John love* is a noun-noun compound. We found characteristic problems in chunking and parsing as well, which we were able to partially correct using simple heuristics.

Once these tagging phases are complete, Qanda's question analysis component uses a set of structural heuristics to identify the following aspects of each question:

- *Anchor:* the object that the answer refers to. The answer may be the anchor, or it may be a property (e.g., length, color) or name of the anchor. The anchor will have a type and supertype from Qanda's (rather simple) ontology, e.g., *PERSON* and *AGENT*. The supertype is used as a backoff for some statistics.
- *Property:* the property, if any, of the anchor that is the actual answer, e.g., the height of a mountain. Properties also have a type and supertype in Qanda's ontology.
- *Name:* the name, if any, of the anchor that is the actual answer. This case can arise in questions that require descriptive answers, as in *Who is Henry Kissinger?*
- *Answer restriction*: an open-domain phrase from the question that describes the anchor, e.g., *first woman in space*.
- *Event*: the main event in the question, if any; typically the main verb, unless it is simply *be*.
- *Salient entity*: What the question is "about". Typically a named entity, this corresponds roughly to the classical notion of topic, e.g., *Matterhorn* in *What is the height of the Matterhorn?*
- *Geographical restriction*: Any phrase that seems to restrict the question's geophysical domain, e.g., *in America*.
- *Temporal restriction*: Any phrase that similarly restricts the relevant time period, e.g., *in the nineteenth century*.
- *Superlative:* Relevant adjectives from the question restriction, e.g., *first*, or *fastest*.

These features are emitted as annotations on the question, and are then available for down-stream components to consume.

3. Answer ranking

Qanda only examines sentences that match the question sufficiently, based on the IDF-weighted overlap described above. It collects candidate answers by gathering phrasal annotations from all of the semantic taggers, and identifies a number of features. These are combined using a conditional maximum-entropy model trained from past TREC QA data sets. Several TREC participants have used this approach, e.g., Ittycheriah et al (2001).

Answer candidate features

Many of the features used in the log-linear model reflect particular kinds of overlap between the question and the context in which the candidate answer is found:

- *Context IDF Overlap*: Described above.
- *Context Unigram Overlap*: Raw count of words[1] in common with the question.
- *Context Bigram Overlap*: Raw count of word bigrams in common with the question.
- *Context Question Restriction Overlap*: Raw count of words from the restriction phrase of the question (see Section 2). Most of the question components described engender analogous overlap features.
- *Context Salient Overlap*: Raw count of words considered especially salient by question analysis (see Section 2).
- *Context Synonym Overlap*: Raw count of words that could be synonymous with questions words.

The synonym features are computed with respect to WordNet (Fellbaum 1998).

Several features are computed based on the candidate itself, or its location in the context sentence:

- *Candidate Overlap*: Raw count of words in common between the candidate itself and the question, to bias against entities from the question being chosen as answers.
- *Candidate Overlap Distance*: Number of characters between the candidate and the closest (content) question word in the context.
- *Candidate Question Restriction Distance*: Number of characters between the candidate and a word from the restriction phrase of the question. Analogous distance features are formed for several other question component phrases.

The only document-level feature currently used is the following:

- *IR Ranking* of the source document by the IR system (but see below).

Candidates with the same textual realizations are merged, with the combined candidate retaining the highest value for each feature. This is the simplest candidate combination possible, but previous work on more robust answer combination across QA systems (Burger & Henderson, 2003) used Tanimoto set distance to compare answer strings as bags of characters. This year we used several cross-candidate features:

- *Merge Count*: (log of) count of identical candidates merged together.
- *Answer similarity*: Average character-level similarity between this candidate and all others.

The latter feature allows textually similar candidates to "vote" for each other. This is especially useful for dates and other types with multiple formats and representations, thus, *January, 1964* and *Jan 64* can support each other without requiring sophisticated coreference.

A number of boolean features are also computed that compare the question's expected answer type with the semantic type of the candidate:

- *Type Same*: True if the candidate and expected answer types are identical.
- *Type Consistent*: True if the candidate's type is "similar" to the expected answer type.
- *Type-Pair*: This is a series of features corresponding to selected pairs of consistent types (see below).

For the most part, candidates are only considered for a question if their types are consistent. For example, *Where* questions lead to an expected answer type of *LOCATION,* which is consistent with *COUNTRY* candidates; *How much* questions lead to *QUANTITY,* consistent with *PERCENTAGE*.

[1]All of the "raw count" features described in this section omit stop words.

Question expected answer type	**Candidate type.**
PERSON	ORGANIZATION
PERSON	COUNTRY
NAME	PERSON
NAME	ORGANIZATION
NAME	LOCATION
CITY	LOCATION
DATE	YEAR
DATE	YEAR
ORGANIZATION	*other*
AMBIGLONG	DURATION
AMBIGLONG	LENGTH
AMBIGBIG	LENGTH
AMBIGFAST	SPEED
MEASURE	MASS
MEASURE	MONEY
MEASURE	MISCMEASURE
MEASURE	*other*
QUANTITY	PERCENT
unknown	LOCATION
unknown	ORGANIZATION
unknown	PERSON

Figure 1: Type-pair features used in evaluating answer candidates

Ideally, Qanda would consider all candidates for all questions, but, if nothing else, performance considerations justify limiting this. We do not even represent all consistent pairs as explicit features. Instead, a small set of approximately 20 combinations was chosen by hand, as indicated in Figure 1. These represent particular biases or preferences that we feel justified in trying to acquire from the training data. In addition, some of these pairwise features represent exceptions to the consistency requirement, e.g., *PERSON* is not consistent with *COUNTRY*, but we wish to consider such candidates anyway. Similarly, we wish to consider certain named entity types as candidates, even when question analysis was unsuccessful in divining an expected answer type (*unknown*).

After all of the (merged) candidates have been acquired, most of the raw feature values described above are normalized with respect to the maximum across all candidates for a particular question, resulting in values between 0 and 1. We have found that features normalized in this way are more commensurate across questions, especially word overlap and related features (Light et al. 2001).

Maximum entropy models

The normalized features are combined using the weights assigned by a maximum entropy model during training. This year, we trained the model using the question sets from TREC 1999 through 2003, including the 2001 list questions, as well as the 25 AQUAINT definition evaluation questions. Last year's questions (TREC 2004) were used as a development set. We used Daumé's (2004) MegaM package to train the models.

We noted some interesting issues in training set conditioning. Because we are using a very small data set, there are arguably too few positive instances to acquire adequate feature weights, especially if we are interested in feature combination. In order to offset this, we experimented with "forcing" Qanda to consider all correct answers (as defined by NIST's judgment sets), even those that it would ordinarily not examine.

A simple example are those correct answers found only in documents that don't make it past Qanda's top-N IR cutoff (typically 50 documents). When we forced these documents into the pipeline (or alternatively increased the IR cutoff, and down-sampled negative candidates), we found that certain features were assigned unintuitive weights by the log-linear model. In particular, the IR rank feature went from a low positive weight to a high negative weight. In retrospect, this makes sense, because high-ranking documents were being over-represented in the positive training instances being presented to the model. In the end, we found it simplest to discard this feature entirely, since it never contributed very much.

4. Definition questions

Qanda has no real facility for processing definition questions as such. Instead, we attempt to leverage our factoid question processing, which for the most part only considers named and other entities as candidate answers. Of course, very few definition answers correspond directly to named entities, per se, but we have noticed that certain kinds of named entities are involved with some definition answers, as indicated in the example below:

Who is Gunter Blobel?

Is at ***Rockefeller University***
1999 Nobel *prize in Medicine*
was born in ***1936***
was born in ***Waltersdorf, Silesia, Germany***

Qanda's question analysis component could already identify the semantic type of the definition target (e.g., *PERSON,* above). Since definition answers did not need to be exact, we allow Qanda to consider certain entity types as pseudo-answers to definition questions.

Previously, Qanda generated the actual definition answer strings by extracting approximately 90 characters around the putative candidate. However, we were struck by BBN's simple, but successful, baseline system from several years ago. This returns entire sentences containing the definition target. Accordingly, in recent years Qanda's definition answers are constructed from entire sentences, as described in the next section.

We used the type-pair features described in Section 3 to license certain combinations of definition target type and candidate type, as shown in Figure 2.

Additionally, we inject some non-entity candidates using crude heuristics for identifying short fragments occurring in appositional contexts. Our hope is that the type-pair features, as well as the candidate count feature, would allow the system to find some definition answers. As training data, we used last year's definition questions, as well as 24 questions from TREC 1999 and 2000 that we determined were essentially definition questions, and the AQUAINT definition questions.

We have had some success with this approach. To illustrate, our best-scoring definition run this year produced the following sentences as a "definition" for *Bollywood* (question 72.7):

SWISS-INDIA-FILMS (Lenk, Switzerland) _ Many of the <u>800 to 900 films produced each year</u> by the Indian movie industry, which is sometimes called Bollywood, feature spectacularly scenic backgrounds that are filmed in faraway locations, typically ***Switzerland****.*
``Often I go past a cinema in London and look at the queues for Bollywood films and there are as many white faces as ***Indians****," he said in a recent interview.*
Television production houses, such as ***Sony Entertainment*** *and* ***Star TV****, pay <u>huge sums to buy the rights of Bollywood favorites</u>.*

Definition target type	**Candidate type.**
PERSON	DATE
PERSON	YEAR
PERSON	PERSON
PERSON	LOCATION
PERSON	COUNTRY
PERSON	*fragment*
ORGANIZATION	LOCATION
ORGANIZATION	COUNTRY
ORGANIZATION	PERSON
ORGANIZATION	*fragment*
unknown	*fragment*

Figure 2: Type-pair features used in evaluating answer candidates

These sentences were chosen based on the following definitional pseudo-answers, which occurred with the indicated counts in sentences containing the definition term *Bollywood*:

- ***Switzerland***: 15 occurrences
- ***Indians***: 14 occurrences
- ***Star TV***: 4 occurrences
- ***Sony Entertainment***: 4 occurrences

The three-sentence answer above matched both of the vital nuggets for *Other* question 72.7, as indicated by underlining. This gave a recall of 1.0. However, because we use entire sentences, the answer is rather long, and provided none of the optional nuggets, so its precision was 0.12, leading to an F of 0.52

5. Final answer generation

Except for the pseudo-answers used for definition questions, most of Qanda's processing is independent of the question type. In particular, list questions are treated entirely as factoid questions until the very last stage, actual answer string generation. Here, special processing is required for both definition and list questions.

In the past, we have simply picked the top N candidate answers, with some fixed cutoff, but in recent years we have attempted something slightly more sophisticated for list and definition questions, picking N dynamically so as to maximize our expected score.

The basic idea takes advantage of Qanda's candidate evaluation mechanism—since this is probabilistic in nature, we can use it to choose how many answers to generate dynamically, based on the expected value of the score we might receive. Both list and definition questions are scored with variants of *F*-measure, the weighted harmonic mean of precision and recall:

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

P is precision, the fraction of our generated answers that are correct, while *R* is recall, the fraction of all possible correct answers that we generated. β is a weight used to place more emphasis on either precision or recall. For list questions, *P* and *R* were weighted evenly, and so the evaluation simply reduces to the following:

$$F_{\text{list}} = \frac{2c}{n + r}$$

Here, n is the number of answers we choose to generate, c is the number of correct answers we generate, and r is the total number of correct answers possible. The evaluation for definition answers was more complicated. The basis of the evaluation done by NIST is not answer strings, but interesting "nuggets" of information. The evaluators attempt to enumerate all correct nuggets by pooling all the system responses—this gives them a value for r. However, it was decided that n is quite difficult to determine—how many nuggets of information, correct or incorrect, are there in a particular text passage? Instead, precision is approximated with a length allowance—each correct nugget is given 100 characters to be expressed. An additional complication with definition answers is that β is set to three. This results in the following expression for evaluating a definition answer set:

$$F_{\text{def}} = \frac{10\hat{P}R}{9\hat{P} + R}$$
$$\hat{P} = \min(1.0,\ \ 100c/l)$$
$$R = c/r$$

Here, c and r are as before, and l is the total length of the answer set, while the min function caps precision at 1.0..[2] With these formulae in hand, we can attempt to estimate the score that a particular set of list or definition answers will receive.

We do not know in advance whether an answer is correct, but we can use Qanda's probabilistic score for the answer candidates as the basis for an expectation of c. We have no real hope of estimating r, the number of correct answers possible, although David Lewis has suggested using the sum of scores over *all* answer candidates for a particular question. We have experimented with this, but found the results to worsen slightly, so for the official TREC runs we simply fix r at a magic number of 5.

Thus, our algorithm for generating list and definition answers is to add each of the candidates to the answer set in turn, increasing n by one each time. Qanda then calculates the expected score of this answer set using the appropriate F-measure variant above, estimating c as follows:

$$c \approx \sum_{i=1}^{n} s_i$$

Here, s_i is the probabilistic score assigned to candidate i. For list questions, we simply add candidate answers in order of their confidence score. For definition questions, as noted above, we decided to use entire sentences as answer set components. Our pseudo-answer candidates, however, are entities and other short phrases. So, we add the matrix sentence of the pseudo-answer to the definition answer set, as described in Section 4. Borrowing another trick from BBN's definition system last year, if any such sentence has too many words in common with the answer set so far (70% or more), we skip it. We do, however, include these skipped answers in calculating c, because a particular sentence may well contain multiple correct nuggets, as in the *James Dean* examples above.

We stop adding candidates to the answer set when the expectation begins to decrease. On last year's list questions, this mechanism performed markedly better than simply generating a single candidate per list question (F = 0.143 vs. 0.060). We have not yet separately evaluated this mechanism on this year's list or definition data.

6. Runs and results

This year we submitted three variant runs. Run A is from a basic configuration, with the features largely as described in Section 3, but with no feature combinations. Run B is a "bells and whistles" run, with substantial feature combination enabled. Run C is the closest to last year's submission, with fewest features. Results are shown in Figure 3. All of our development this year centered on factoid questions, but, surprisingly, run C performed best on those questions, while run B performed best on definitions. We have yet to explain this adequately.

Run	Factoid	List	Definition
A	0.113	0.060	0.167
B	0.116	**0.080**	**0.217**
C	**0.180**	0.047	0.032
Median	0.152	0.053	0.156

Figure 3: Results for three MITRE runs compared to the 2005 medians

[2] In fact, this evaluation metric was even more complicated, as the assessors made a distinction between inessential and essential correct nuggets—only the latter counted for recall. We declined to attempt to estimate the essentialness of an answer.

7. Conclusion

As well as the usual description of this year's system architecture, we have discussed Qanda's question analysis and our use of maximum entropy models for answer selection. We also presented our approach to generating definition and list answers using essentially the same system as for factoid questions, as well as the mechanism we use to determine how many of these answers to provide.

References

Apache Software Foundation, 2002. "Jakarta Lucene—Overview". http://jakarta.apache.org/lucene/.

John D. Burger, John C. Henderson, William T. Morgan, 2002. "Statistical named entity recognizer adaptation", in *Proceedings of the Conference on Natural Language Learning.* Taipei.

John Burger, John Henderson, 2003. "Exploiting diversity for answering questions", in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

John D. Burger, Scott Mardis, 2002. "Qanda and the Catalyst architecture", in *AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*.

Hal Daumé III, 2004. "Notes on CG and LM-BFGS optimization of logistic regression". Unpublished. http://www.isi.edu/~hdaume/megam/

Christiane Fellbaum, ed., 1998. *WordNet: An Electronic Lexical Database*. MIT Press.

D. Ferrucci and A. Lally, 2004. "Building an example application with the Unstructured Information Management Architecture." *IBM Systems Journal* 43:3.

Abraham Ittycheriah, Martin Franz, Salim Roukos, 2001. "IBM's statistical question answering system", in *Proceedings of the Tenth Text REtrieval Conference (TREC-10)*. NIST Special Publication 500-250.

Marc Light, Gideon S. Mann, Ellen Riloff, Eric Breck, 2001. "Analyses for elucidating current question answering technology", in *Natural Language Engineering* 7(4).

Guido Minnen, John Carroll and Darren Pearce, 2001. "Applied morphological processing of English". *Natural Language Engineering*, 7(3).

Eric Nyberg, John D. Burger, Scott Mardis, David Ferrucci, 2004. "Software architectures for advanced question answering", in *New Directions in Question Answering*, ed. Mark Maybury. AAAI Press.

Adwait Ratnaparkhi, 1996. "A maximum entropy part-of-speech tagger," in *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

Gerold Schneider, Fabio Rinaldi, James Dowdall, 2004. "Fast, deep-linguistic statistical dependency parsing". Workshop on Recent Advances in Dependency Grammar, COLING 2004, Geneva.

Ben Wellner, 2005. "Carafe". Unpublished. http://sourceforge.net/projects/carafe/.