MITRE's Qanda at TREC-15

John D. Burger
The MITRE Corporation
john@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 six person-weeks this year. (Accordingly, much of this discussion is plagiarized from prior system descriptions.) We made a number of small improvements to the system this year, including expanding our use of Wordnet. The system's information retrieval wrapper now performs iterative query relaxation in order to improve document retrieval. We also experimented with an ad hoc means of "boosting" the maximum entropy model used to score candidate answers in order to improve its ranking ability.

1. TREC-15 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 interactions between language processors, based on the Catalyst infrastructure used in previous versions of Qanda (Burger 2004, Burger & Mardis 2002, Nyberg et al. 2004). 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.

Major system components

Qanda has a by now banal QA architecture, which proceeds in several phases. Questions are analyzed for expected answer types, 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. Qanda is composed of several dozen components connected via AMS—below we describe each of the major phases in turn.

- *Common question and document processing*: This consists of several steps: tokenization, sentence boundary detection, part of speech tagging (Ratnaparkhi 1996), morphological analysis (Minnen et al. 2001), fixed phrase tagging (see below), and entity tagging. For the latter, we use Phrag (Burger et al. 2002), an HMM-based tagger, to identify named persons, locations and organizations, as well as temporal expressions.
- *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. (Retrieved documents do *not* have this level of analysis applied to them.)
- *Document retrieval*: AMS components have been written for several IR engines. These take the results of the question analysis and formulate a series of queries for each question—Section 3 explains this in more detail.
- *Passage processing*: After the retrieved documents pass through the common analysis phase, Qanda identifies lexical relations between the words in each sentence and those in the question (see Section 4). It also assigns a preliminary score to each sentence by summing the log-IDF (inverse document frequency) of words common to the sentence and the question. Those sentences with a low score are not processed by most of the system, improving the efficiency of more expensive downstream components.
- *Fixed repertoire taggers*: In addition to named entity tagging, we have a simple facility for constructing AMS taggers from fixed word- and phrase-lists. Some of these are used in question analysis to help determine the expected answer type. Others 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 (although these latter are less relevant to the more recent TREC QA incarnations).

- *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).
- *Answer collection and ranking*: Candidates are identified and merged, a number of features are collected, and a score is computed (Section 4).
- *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 5.

All of these components communicate by consuming and producing stand-off annotations via AMS. 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 appropriately.

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. As of last year, the system includes a detailed parsing phase using MITRE's conditional random field chunker Carafe (Wellner, 2005) and the Pro3Gres dependency parser from the University of Zurich (Schneider et al. 2004).

Perhaps due to the scarcity of questions in standard corpora, 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 trained on declarative data to attempt to tag questions like *Who does John love?* as if *John love* is a noun-noun compound. We find analogous problems in chunking and parsing as well, which we are able to correct to some extent 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 which 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*.
- *Superlative:* Relevant adjectives from the question restriction, e.g., *first*, or *fastest*.
- *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* and *temporal restriction*: Phrases that can be interpreted as restricting the question's geophysical domain, or time period, e.g., *in America*, or *in the nineteenth century*, respectively.

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

3. Document retrieval

The results of question analysis are passed to a document retrieval component—for TREC this was an AMS wrapper around the Java-based Lucene engine (Apache 2002). This formulates an IR query from the question components described above with the goal of retrieving documents likely to answer the question. An ideal document would contain every term from the question, in fact, every question component as a full phrase. This is, of course, usually too much to hope

for, so Qanda begins with a restrictive query, but then relaxes it until a target number of documents is retrieved, typically 50. Question components are relaxed in a heuristic order that we arrived at through trial and error, and new documents are added to the retrieval set until the target number is reached. This is similar to the first "feedback cycle" performed by some of LCC's TREC QA systems (Moldovan et al. 2002).

For example, for question 143.4:

> *Who is the senior vice president of the American Enterprise Institute ?*

Qanda formulates the following initial query:

> *+"senior vice president" +"American Enterprise Institute"*

In Lucene's query syntax, the quotes indicate that only the complete phrases should match, while the plus signs require the phrases to be present. Such a query retrieves only documents containing both exact phrases—there are apparently only four such documents in the AQUAINT collection, so another round of retrieval is attempted, with a weaker query:

> *senior vice president "senior vice president" +"American Enterprise Institute"*

This query still requires the topic phrase, as that is deemed most important, but has weakened the requirements on the other phrase—the words can appear individually, and in fact none of the words need appear in a retrieved document. The entire phrase is still retained as optional, as Lucene will more highly rank documents containing the phrase.

The second, weakened query retrieves 350 documents, and the system greedily adds novel documents to the result set from the original query. When the target number of documents is reached, the cycle of query relaxation stops, and the documents are passed to the rest of the system. In this case, if necessary, a fully weakened query would have been used:

> *senior vice president "senior vice president" American Enterprise Institute "American Enterprise Institute"*

Lucene's query syntax also allows weighted terms, and in actuality, Qanda uses different weights for each question component, even when weakened. Like the relaxation order, these weights were arrived at heuristically, and only a small portion of the full query space is explored by Qanda. Ideally, we would like to automatically acquire the weights, relaxation order, etc., so as to optimally traverse this space.

4. Answer ranking

The retrieved documents are then examined by a number of taggers and other processors. As indicated in the overview, most components of Qanda skip sentences that do not sufficiently match the question, based on an IDF-weighted overlap threshold. This year we lowered this threshold substantially so that, effectively, every sentence containing at least one content word from the question is fully processed. Qanda collects candidate answers by gathering phrasal annotations from all of the semantic taggers, and identifies a number of features for each candidate. 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 maxent 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 and Bigram Overlaps*: Raw counts of words/bigrams[1] in common with the question.
- *Context Question Component Overlaps*: Raw counts of words from various components of the question (see Section 2).
- *Context Wordnet Overlap*: Raw counts of words that could be synonyms, hypernyms, etc. of questions words. Count features for most Wordnet relations (Fellbaum 1998) are used.

A number of 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 Component Distances*: Number of characters between the candidate and various components of the question found in the candidate context.

[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

Candidates from the same document with the same textual realizations are merged, with the combined candidate retaining the best value for each feature. This is the extent of Qanda's candidate combination—no coreference is currently performed. We use 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, allowing, for example, *January, 1964* and *Jan 64* to support each other without requiring any explicit coreference. We have also used such an approach to combine answers from multiple QA systems (Burger & Henderson, 2003).

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 *LOCATION, CITY* and *COUNTRY* candidates; *How much* questions lead to *QUANTITY,* consistent with *PERCENTAGE*.

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, we use a small set of approximately 20 combinations 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, as the latter can sometimes answer questions such *Who started the six-day war?* 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 previously found that features normalized in this way are more commensurate across questions (Light et al. 2001). This year we also explored unit Gaussian normalization, as well as quantile normalization, but found results to be inconclusive. All of our official TREC runs simply used max-value-per-question normalization.

Maximum entropy models

The normalized features are combined using the weights assigned by a maxent model during training. This year, we trained the model using the question sets from TREC 1999 through 2004, including the 2001 list questions and the 25 AQUAINT definition evaluation questions. Last year's questions (TREC 2005) were used as a development set, although for our final run (MITRE2006D) we included this development data in the training. We used the MegaM package (Daumé 2004) to train these models.

In previous years we have struggled with a number of issues involving training this part of the system, for

example, the question of how to normalize feature values discussed above. Other issues arise because we are using a fairly small data set—there are arguably too few positive instances to acquire adequate feature weights, especially if we are interested in feature combination (our training data s typically 98–99% negative instances). Last year, we experimented with forcing Qanda to consider *all* correct answers (as defined by NIST's judgment sets) during training, even those the system would ordinarily not examine, but ultimately found this to produce inferior results.

This year we spent some time exploring the problems involved in using a discriminative model such as maxent to rank candidates. We encountered a number of cases of feature additions or other modification to the system that decreased the maxent model's estimation error, but had negative effects on its ranking ability. This is frustrating, but perhaps understandable—MegaM is choosing feature weights to maximize the likelihood of all of the data, but we are in fact only interested in the candidate that ranks highest using the resulting probability estimate.

For many questions, there are multiple correct answer-document pairs—some are harder than others for the model to "justify". In a sense, we would like the model to focus on those correct answers that it can more easily rank highly, possibly at the expense of other correct, but more "difficult" candidates, in the context of a particular question. MegaM in fact has a simply facility for weighting some instances more than others, and we used this to perform a crude form of "answer boosting". Our procedure is thus: we first train a maxent model from the training data, and then use this to rank all of the training answers for each question. Then, within a question, we weight the top *n* instances with a weight *w* (greater than 1), and retrain the model. This second model is then used at runtime.

Again, the intuition is to force the model to focus on those correct candidates that can already be ranked fairly highly. Similarly, we wish to focus on those incorrect candidates that are confusable with the best correct candidates. We are willing to "sacrifice" poorly ranked candidates if at least one correct candidate per question can be ranked highly. We typically used values of 100 and 2 for *n* and *w*, respectively, and found that this ad hoc procedure could consistently improve our factoid scores by roughly ten percent, relative. Our TREC runs, A and C differ only in whether this boosting was performed (see Figure 3).

5. Definition questions

Qanda has no real facility for processing definition (*other*) questions as such. Instead, we 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 were 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

In a sense, while named entities alone might not constitute good definition nuggets, they form the "kernel" of many nuggets. Qanda's question analysis component can already identify the semantic type of the definition target (e.g., *PERSON,* above). Since definition answers do not need to be exact, we allow Qanda to consider certain entity types as pseudo-answers to definition questions. Then, at the end, the actual definition text is constructed from the matrix sentences in which these pseudo-answers are found (see Section 6).

We used the type-pair features described in Section 4 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, allow the system to find some definition answers. As training data, we use the explicit *other* questions from recent TRECs, the AQUAINT

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 definition pseudo-answers

definition questions, and a number of questions from previous years that we determined were essentially definition questions.

6. Final answer generation

Most of Qanda's processing is independent of whether the question is factoid, list, or other. One exception is the fragment pseudo-answers generated for definition questions, another is that the question type is, in fact, available as a feature to the maxent scoring model. Otherwise, however, the system performs the same processing on all questions, until the very last stage, actual answer string generation. Special processing is required to generate both definition (other) and list answers.

List questions generate a set of short (factoid) answers, while definition questions are a set of full sentences containing the candidate pseudo-answers described above. Earlier versions of Qanda simply picked the top n candidate answers, with fixed cutoffs for list and definitions. For several years, we have used something slightly more sophisticated—the system determines this cutoff dynamically so as to maximize the expected score for each question.

The basic idea takes advantage of Qanda's candidate evaluation mechanism—since the maxent model produces probability estimates for the correctness of each individual answer, we can use these to reason about the expected value of the score an entire answer *set* might receive. Our algorithm for generating list and definition answers is thus to greedily add each of the ranked candidates to an answer set in turn, stopping when the expected score appears to decrease.

The expected score of an answer set of course depends on the scoring metric to be used. Both list and definition questions are scored with variants of *F*-measure, the weighted harmonic mean of precision and recall:

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

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, NIST weighted P and R evenly, and so the evaluation simply reduces to the following:

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

For definition questions, β is set to three, and precision is approximated with a length penalty:

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

In both F_{def} and F_{list}, c is the number of correct answers—either the number of distinct list answers judged correct, or the number of correct nuggets found in a definition answer. r is the total number of correct answers possible, according to the NIST assessors. For list questions, n is the total number of answers generated, and for definition questions, l is the total length of an answer.[2]

Based on these equations, Qanda can estimate the expected score of an answer set. We estimate c as the sum of the maxent scores for the answers in the set. It remains to estimate r, the number of correct answers possible. This is, of course, difficult,[3] and so we simply use a fixed value, in this case the means from last year's data.

$$r_{\text{list}} = 12$$
$$r_{\text{def}} = 4$$

For list questions, we add each factoid-style answer to the answer set in turn, incrementing n by 1 with each, as long as F_{list} increases. Similarly, for definition questions, we add each pseudo-answers matrix sentence to the answer set, incrementing the length l appropriately, as long as F_{def} continues to increase.

7. Submitted runs and results

This year we submitted three variant runs (see Figure 3). Run A is from a basic system, with the features described above. Run C differs only in that the candidate scoring model has been "boosted" as described in Section 4—the roughly ten percent improvement is similar to what we saw with our development data. It is unclear why the evaluation list questions appear to be unaffected by the boosting. Run D differs mainly in that our development set has been added to the training set, namely, last year's questions. This did not include, however, the

[2]Qanda ignores the distinction between inessential and essential correct nuggets.

[3]David Lewis (personal communication) has suggested using the sum of scores over *all* answer candidates for a particular question as an estimate for r, but we have found this to worsen our results.

Run	Factoid	List	Other	Overall
A	0.181	0.083	0.136	0.130
C	**0.208**	0.083	**0.156**	**0.149**
D	**0.208**	**0.087**	0.131	0.139
Median	0.186	0.027	0.125	0.134
Best	0.578	0.433	0.250	0.394

Figure 3: Results for three MITRE runs, as well as median and best across all 2006 submissions

definition questions, perhaps accounting for run D's loss in this area.

8. 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, in particular a method for boosting such models to better support TREC's winner-take-all evaluation. We believe that this ad hoc method has some connections to minimum-risk annealing (Smith & Eisner, 2006) and would like to explore this in the future. We described the query relaxation technique that Qanda uses in an attempt to improve the document retrieval phase of the system—we would like to use a more principled exploration of the query space for retrieval. Finally, we presented our approach to generating definition and list answer sets by maximizing the expected score each set will receive.

References

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

Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais, Andrew Ng, 2001. "Data-intensive question answering". in *Proceedings of the Tenth Text REtrieval Conference (TREC-10)*. NIST Special Publication 500-250

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.

Smith, David A. and Jason Eisner (2006). "Minimum-risk annealing for training log-linear models", in *Proceedings of the International Conference on Computational Linguistics and the Association for Computational Linguistics* (COLING-ACL)

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).

Dan Moldovan, Marius Paca, Sanda Harabagiu, Mihai Surdeanu, 2002. "Performance issues and error analysis in an open-domain question answering system", in *Proceedings of the 40th Annual Meeting of ACL*.

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 and Marc Vilain, 2006. "Leveraging machine readable dictionaries in discriminative sequence models", in *Proceedings of Language Resources and Evaluation Conference* (LREC).