Concordia University at the TREC-15 QA track

Leila Kosseim, Alex Beaudoin, Abolfazl Keighbadi and Majid Razmara
CLaC Laboratory
Department of Computer Science and Software Engineering
Concordia University
1455 de Maisonneuve Blvd. West
Montreal, Quebec, Canada, H3G 1M8
kosseim,aaw_beau,a_keigho,m_razma@cse.concordia.ca

Abstract

In this paper, we describe the system we used for the TREC Question Answering Track. For factoid and list questions two different approaches were exploited: A redundancy-based approach using a modified version of ARANEA and a parse-tree based unifier. The modified version of ARANEA essentially uses Google snippets for extracting answers and then projects them to AQUAINT. The parse-tree based unifier is a linguistic-based approach that chunks candidate sentences syntactically and uses a heuristic measure to compute the similarity of each chunk in a candidate to its counterpart in the question. To answer *other* types of questions, our system extracts from Wikipedia articles a list of *interest-marking* terms related to the topic and uses them to extract and score sentences from the AQUAINT document collection using various interest-marking triggers.

We submitted 3 runs using different variations of the system. In the factoid run, the average of our 3 runs is 0.202, for the list, we achieved an average of 0.084, and for the "Other", we achieved an average F-score of 0.192.

1 Introduction

This is the first year in a while that Concordia University participated in TREC-QA. We only participated in the main task, which consisted in three types of questions: factoid, list and *other*. Although all three question types contributed equally in the final system score, we put most of our efforts on factoid and *other* questions, and spend very little time on list questions.

As our old system [1] received poor results in 2002 and had not been improved since then, we decided to take advantage of the freely available ARANEA system[1], successor of AskMSR [2, 3]. For factoid questions, we used two main approaches: a redundancy-based QA system working on the Web (a modified version of

[1]http://www.umiacs.umd.edu/~jimmylin/downloads/index.html

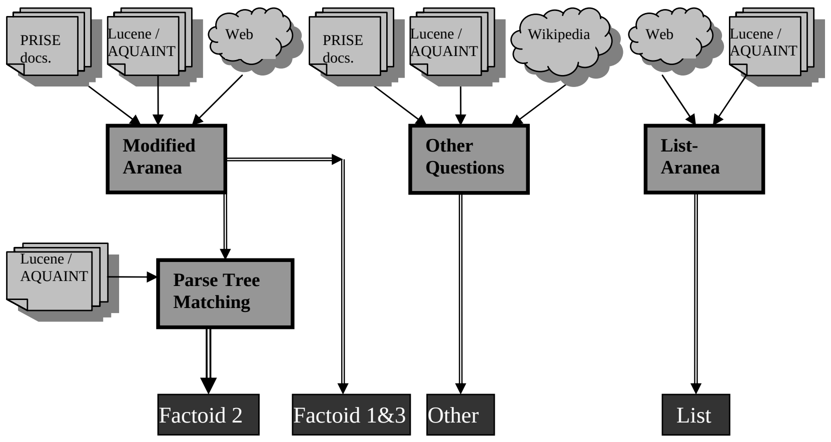

Figure 1: Overall Architecture of QASCU

ARANEA) and a linguistic-based system working on AQUAINT only. We spend about 3 person-month improving ARANEA and modifying it for the 2005 task specifications. In parallel, we developed our own QA module that does a more fine-grained analysis of candidate sentences based on the parse tree similarity between the questions and candidate sentences.

For the *Other* questions, we used terms extracted from Wikipedia and projected them on the AQUAINT collection. Sentences containing these Wikipedia terms are then ranked using various interest-marking triggers.

In the following sections, we describe our approach to answer factoid, list and other questions. Finally, we present our results in section 3.

2 The Overall Architecture

The overall architecture of our system, called QASCU, is shown in Figure 1. The three main modules are shown: the linguistic module based on parse tree unification, the modified ARANEA module, and the *other* module. The list-question module is also shown in the figure, but as mentioned previously, very little work was done on this module.

2.1 Answering Factoid Questions

Our approach for answering factoid questions is a combination of:

1. a modified version of Jimmy Lin's ARANEA system, and
2. a linguistic parse-tree unification algorithm.

2.1.1 Modified Aranea

ARANEA is a web-based rule and statistical QA system. A simplistic overview of the system is that it generates queries and question reformulations, gets snippets from Google and Teoma, generates ngrams as possible answers, collates the ngrams based on n-gram similarity, does filtering based on the expected type of answer and number of supporting documents, reranks candidates based on frequency of words within AQUAINT, then projects the answer onto the AQUAINT corpus. Detailed descriptions of ARANEA can be found in [4, 5, 6].

Several changes have been brought to the original ARANEA system. These include: reformulating the web query, adding new data sources, predicting answer types and projecting the answers onto the AQUAINT corpus.

ARANEA uses snippets returned from Google and Teoma for its IR. Teoma was dropped since it was bought by Ask.com and the web interface changed. The system was originally written before TREC included the target based questions. Initially, we opted to solve this by simply adding the target to our query. Unfortunately, this caused the query size to balloon. Since Google has a hard limit of 10 words in its query, we chose to put the target at the beginning of the query and only keep the nouns, adjectives, verbs and numbers from the questions into the query. One noticeable source of error was that ARANEA would not find a correct answer at all, nearly half the time. Therefore, in addition to the Google snippets, we used two new data sources:

- AQUAINT – Lucene[2] was used to search for relevant articles from the AQUAINT document collection using the same query sent to Google.
- GoogleText – In addition, also we downloaded the actual web pages found by Google (hereto known as GoogleText), as opposed to searching only the Google snippet. Not all Google documents were downloaded since large, unfocused articles are quite common and we do not want to add too much noise, therefore we chose to only include those who were smaller than 30K.

All these documents were broken into sentences and ranked by their word overlap with the question.

Another noticeable source of error was that ARANEA would often return answers that were not of the correct type. For example, returning a location

[2]`http://lucene.apache.org`

instead of a person. The QASCU system generates an expected type (Number, Date, Person, Organization, Location, Unknown) through a simple word-match and ordering algorithm which was generated by hand from the 2005 question set, then uses the GATE NE [7] tagger, with a few extras for Unknown and Numbers, to return candidates that are then scored by their relevance to the type of answer expected. These candidates are then used instead of the snippets returned by Google, Lucene or GoogleText. For Numbers, all numbers and following noun phrases are returned and for Unknown type questions, noun phrases and capitalized entities (strings of capitalized words, including internal prepositions (e.g.: *Unites States of America*)) are returned.

Because ARANEA is fundamentally based on answer redundancy, it would often give the most common piece of information of the correct type even though it may not answer the specified question (e.g: a question on *Kurt Cobain*'s birth will return the date of his death since there are so many more texts that cover his death than his birth.) To diminish this, instead of using the frequency over the whole of the AQUAINT corpus, we rerank using the frequency over the top 50 documents from PRISE.

Since ARANEA is a web based solution, the answer must be projected onto the corpus. This projection is done quite naively by increasing the number of documents returned by Lucene and searching through them when searching for supporting documents. If no document is found, then a subset of the answer is searched for depending on the answer type.

With these modifications, we managed to double the performance of ARANEA on the 2005 TREC data (from an accuracy of 0.170 to 0.321).

2.1.2 The Parse-Tree Unifier

In addition to the modified ARANEA, we also developed a parse-tree matching algorithm to identify and rank candidate sentences from the AQUAINT collection with the given question based on syntax.

The unifier performs its own information retrieval on the AQUAINT collection only. For this, we used Lucene. To build the query for Lucene, we send the original question+target to a part-of-speech tagger and only keep open class words. The keywords are then weighted according to their parts-of-speech: proper nouns are given a higher weight, then nouns, then verbs, then adjectives and finally, adverbs. The query is passed through Lucene to retrieve a list of relevant documents. If no document is found, the least valuable keyword is dropped from the query and IR is tried again until at least one document is returned. The system then tokenizes each document and marks the sentences that include at least $LIBERTY\_RATIO$ percent of the keywords (this value is set to 60% but experiments show insensitivity of the unifier's accuracy to this value). The semantic relatedness of the question's main verb to each verb in the candidate sentences is verified in the next step. To do so, we use Leacock and Chodorow's similarity measure from WordNet::Similarity [8]. Finally, the arguments are the matched verbs are mapped on each other and the similarity between them is computed. Candidate sentences are then ranked by the extracted similarity values: their verb relatedness to the questions's verb, and the unification score of their arguments.

Once the sentences are ranked, we try to match their parse-tree to the parse-tree of the question. Here, we use the Minipar parser [9] and employ a fuzzy unification method. We observed that starting and limiting the parse-tree unification method from the most similar verb to the question's main verb does not always lead to the correct position in the candidate parse tree; a strong verb similarity must co-occur with an entity match in the subtree. This suggests that a stronger seed point is the root of the subtree that contains the question's head noun phrase.

To choose the question head, we rank all noun phrases in the question and pick the one that contains the most valuable question keywords. If this head phrase is found in the candidate sentence, it will become an anchor to find the relevant verb: we move up from this noun phrase in the parse tree to reach the first parent verb. In long candidate sentence, using an anchor reduces the candidate verbs to the ones that include the question head (or a reference to it).

After a candidate subtree is chosen based on the semantic similarity of its verb, we proceed with the unification by checking whether the selected verb relates the same entities as the question's main verb. A heuristic method evaluates how similar the two subject subtrees are (likewise for object or modifier) subtrees if any). In other words, the object and subject arguments are strong constraints, while the other arguments (such as modifiers) are used to score the accepted sentences. These similarity scores add up to produce the final score of the candidate:

$$Score = \Sigma_{i:subtree} Score(Question_i, Candidate_i)$$

To unify two phrases (subject, object or modifier subtrees) marked by the linguistic method as the arguments of verbs, we apply a heuristic process. This step uses two measures: the number of overlapping words based on a bag-of-words approach and the number of overlapping links. More formally, we compute

$$\alpha \times WordOverlap + (1-\alpha) \times \ LinkOverlap$$

as the total unification score of a sentence. The parameter α shows the relative importance of the two features: in our configuration, we use $\alpha = \frac{1}{3}$, which considers the link-overlap feature to be twice as important as the bag-of-words feature.

The reason we relax our linguistic constraints at this stage is that we are focusing on a sentence that conveys a similar event or state to the question; only a clue about similarity of its verb arguments is sufficient to conclude that its verb is affecting the same entities as the question. Syntactic differences of verb arguments (subtrees) should not critically affect our judgment. We reject a candidate if its arguments have no keyword based overlap with the question's.

the entities might probably be mentioned in syntactically different phrases.

Since the unifier identifies sentences (as opposed to exact answers), we use the unifier to semantically validate and to find a supporting document for ARANEA's answers.

We used the output of ARANEA on the TREC data to improve our IR result. We added m documents from the IR result for the extended query $AraneaAnswers \bigwedge QuestionKeywords$ to the regular document list we get for the original $QuestionKeywords$ query. The expected answer type is also taken from RANEA's output.

The top-most ARANEA candidate is searched in the unifier's answers. If it is found, then this answer is confirmed and the document that this sentence is extracted from is returned as the supporting document. However, if no unifier candidate contains the top answer, the validation process is tried for the top two ARANEA answer and so on until the unifier confirms that answer.

2.2 Answering Other Questions

To answer "Other" questions, we used a notion of *interest marking terms*. Fundamentally, we hypothesized that interesting nuggets can be extracted using two types of interest markers:

1. target-specific interest marking terms (e.g. *Titanic* ⇒ *White Star Line*, *J.F. Kennedy* ⇒ *Lee Harvey Osward*), and
2. universal interest marking terms (e.g. *first man on the moon*, *150 people died*)

To identify target-specific interest marking terms, we used the Wikipedia[3] online dictionary. Articles in Wikipedia relate to many target types and the content of each article is a short summary that highlights the most interesting facts – precisely what we are looking for to find target-specific interest markers.

The first stage to answering *Other* questions is to find the proper Wikipedia article. We use the Google API to search in the Wikipedia domain using the target as query. The first Wikipedia article that satisfies the query is taken. However, if no Wikipage satisfies the query, then we try to loosen the query and eventually if still no Wikipage is not found, the top N AQUAINT documents are used for term extraction.

After the Wikipage or top N AQUAINT documents are retrieved, we extract named entities as interesting terms for each target, and we search AQUAINT for the N most relevant documents. These documents are retrieved by Lucene using the same query generated for the target as in the Wikipage search and a secondary query from the title of the Wikipage if it has been found. This secondary query is ORed to the Google query.

[3]`http://en.wikipedia.org`

Within the documents chosen as the domain, the frequency of each interest marking term is then computed. For each term, we compute a weight as the logarithm of its frequency.

$$Weight(T_i) = Log(Frequency(T_i))$$

All sentences from the domain documents are then scored according to how many target-specific interesting terms it contains. This is computed as the sum of the weight of the interesting terms it contains.

$$Score(S_i) = \sum_{j=1}^{n} Weight(T_j) \quad | \, T_j \in S_i \quad \forall 1 \leq j \leq n$$

After scoring the sentences and throwing away those with a score of zero (i.e. no interesting term in the sentence), we try to remove paraphrases. In order not to remove false paraphrases, we play it conservatively, and only remove lexically similar sentences. Either the sentences are almost equivalent to each other at the string level or they share similar words but not the same syntax. To compare sentences, we have used the *SecondString* package[4], an open-source Java-based package of approximate string-matching techniques [10].

Once the sentences are ranked based on target-specific interesting terms, we boost the score of sentences that contain terms that generally mark interesting information regardless of the topic. Such markers were determined empirically by analyzing the previous TREC data. These markers consists of superlatives, numeral and target-type specific keywords. This last type of marker is essentially a list of terms that do not fit any specific grammatical category, but just happen to be more frequent in interesting nuggets. To identify these terms, we analyzed the data of the 2005 *other* questions and identified, for each type of target (person, organization, ...) a list of terms that are more frequent in interesting nuggets. The score of any sentence that contains a superlative, a numeral or a interesting-term is boosted by 20% per term. Finally, the top N sentences making up 7000 non-white-space characters are returned as our nuggets.

2.3 Answering List Questions

Although list questions accounted for a third of the overall score, we spent very little time on them (about 3 days/person). List are answered by returning the entire list of possible answers returned by the factoid module with two small differences. The list questions differ by requiring fewer supporting documents and that since ”Unknown” answer-types return a lot of noise, only the top 50% of those are returned. Work has been done to increase the accuracy of the answers but this actually lowers the score. The reason for this is that, based on the TREC 2005 data, 56% of the list score comes from the best 9 answered

[4] `http://secondstring.sourceforge.net`

list questions (10%). These tend to be lists with few elements. Unfortunately, when removing results to increase accuracy these few questions tend to do more poorly and thereby lower the final score.

3 Results

For TREC-2006 we submitted three runs. The factoid versions of the system are:

QASCU1-factoid: Modified version of ARANEA alone. We drop anything without at least two supporting documents taken from Google, Lucene, or GoogleText, and we rerank the remaining using the whole AQUAINT corpus.

QASCU2-factoid: Same as QASCU1-factoid + the parse tree matcher.

QASCU3-factoid: Same as QASCU1-factoid, but we drop anything without at least three supporting documents, and we rerank the remaining using the top 50 PRISE documents on the target.

The *Other* section of our systems have also three different submissions:

QASCU1-other: Taking the intersection of top 25 documents from PRISE and Lucene/AQUAINT as the domain documents.

QASCU2-other: Taking the top 50 Lucene/AQUAINT documents as the domain and adding two documents from AQUAINT to the Wikipage for term extraction.

QASCU3-other: Using top 50 Lucene/AQUAINT documents as the domain.

The list part of the 3 runs we submitted consisted of:

QASCU1-list: Less accurate algorithm for “Unknown” type questions.

QASCU2-list: Require only one supporting document.

QASCU3-list: Require at least two supporting documents.

Table 1 shows the official evaluation details of our 3 runs along with the median score of all systems. The results of all our runs are above the median. For the factoid runs, `QASCU3` was significantly better than `QASCU2` and `QASCU1`. `QASCU2` (the run with the parse tree unifier) is better than `QASCU1` (without the parse tree unifier), but since `QASCU3` did not include the unifier and performed much better than the other two, we are not convinced of the unifier’s usefulness, when run independently. Most of the performance seems attributable to the modified ARANEA.

	`QASCU1`	`QASCU2`	`QASCU3`	median
Factoid	0.194	0.199	0.213	0.186
incorrect	272	270	260	
unsupported	20	22	29	
inexact	31	29	25	
locally correct	2	2	3	
globally correct	78	80	86	
List	0.094	0.096	0.063	0.087
Other (F-score)	0.197	0.180	0.199	0.125
Other (pyramid)	0.206	0.194	0.203	0.139

Table 1: Official results of the 3 runs.

Surprisingly, our list runs also performed better than the median; even though we invested so little effort on it. Since the list answers are derived directly from the output of the factoid runs, our performance on the lists is probably a side-effect of our performance on the factoid.

Finally, our F-score and pyramid score on the *other* questions are also higher than the median. Our best runs are `QASCU1` and `QASCU2` which do not use AQUAINT for term extraction. This seems to confirm our intuition that AQUAINT should not be used for term extraction, and Wikepedia, or another source is more appropriate.

4 Conclusion

In this paper, we described the 3 runs we submitted to the QA track. The factoid run is based on a modified version of ARANEA coupled with a syntactic tree unifier. Our *other* run is based on terms found in Wikepedia entries and ranking of nuggets is done through the use of target-specific and target-independent interest markers. Finally, our list run is based on a very crude processing of the output of the factoid runs.

Acknowledgments

First and foremost, we need to thank Jimmy Lin for making ARANEA available for download. Sharing resources allows smaller teams to contribute to research rather than play catch-up.

This project was financially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Bell University Laboratories (BUL).

References

[1] L. Plamondon, G. Lapalme, and L. Kosseim. "the quantum question-answering system at trec-11". In *Proceedings of the 11th Text Retrieval Conference (TREC-11)*, Gaithersburg, USA, November 2002. TREC-11.

[2] M. Banko. AskMSR: Question answering using the Worldwide Web. In *Proceedings of EMNLP 2002*, 2002.

[3] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Proceedings of EMNLP 2002*, 2002.

[4] Boris Katz, Jimmy Lin, Daniel Loreto, Wesley Hildebrandt, Matthew Bilotti, Sue Felshin, Aaron Fernandes, Gregory Marton, and Federico Mora. Integrating web-based and corpus-based techniques for question answering. In *Proceedings of the 12th Text Retrieval Conference (TREC-12)*, 2003.

[5] Boris Katz and Jimmy Lin. Question answering from the web using knowledge annotation and knowledge mining techniques. In *Proceedings of the twelfth international conference on Information and knowledge management (CIKM)*, 2003.

[6] Boris Katz, Gregory Marton, Jimmy Lin, Aaron Fernandes, and Stefanie Tellex. Extracting answers from the web using knowledge annotation and knowledge mining techniques. In *Proceedings of the 11th Text Retrieval Conference (TREC-11)*, 2002.

[7] H. Cunningham. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36:223–254, 2002.

[8] Ted Pedersen, Siddharth Patwardhan, and Jason Michelizzi. Wordnet::similarity - measuring the relatedness of concepts. In *Proceedings of NAACL-04*, 2004.

[9] Dekang Lin. Principle-based parsing without overgeneration. In *Proceedings of ACL-93*, pages 112–120.

[10] W. W. Cohen, P. Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of the IJCAI Workshop on Information Integration on the Web (IIWeb), pages 73-78*, Acapulco, Mexico, 2003.