Question Answering with QED and Wee at TREC-2004

Kisuh Ahn, Johan Bos, Stephen Clark, James R. Curran[*]
Tiphaine Dalmas, Jochen L. Leidner, Matthew B. Smillie, Bonnie Webber

School of Informatics, University of Edinburgh
[*]School of Information Technologies, University of Sydney
trec-qa@inf.ed.ac.uk

Abstract

This report describes the experiments of the University of Edinburgh and the University of Sydney at the TREC-2004 question answering evaluation exercise. Our system combines two approaches: one with deep linguistic analysis using IR on the AQUAINT corpus applied to answer extraction from text passages, and one with a shallow linguistic analysis and shallow inference applied to a large set of snippets retrieved from the web. The results of our experiments support the following claims: (1) Web-based IR is a good alternative to "traditional" IR; and (2) deep linguistic analysis improves quality of exact answers.

1 Introduction

In this report we describe the TREC-2004 entry of the Universities of Edinburgh and Sydney for the question-answering evaluation exercise. This year we experimented with two complementary QA streams: our **QED** system developed in previous years (Leidner et al., 2004), using traditional IR and deep linguistic processing (see Figure 1), and **Wee**, a system developed by Tiphaine Dalmas, using Google and shallow linguistic processing. We were interested in comparing the performances of these two streams, as well as finding out whether they could be successfully combined. We therefore aimed to submit three runs:

- Run A: Wee
- Run B: hybrid Wee and QED
- Run C: QED

In the remaining of this paper we will first describe the two systems in detail: Section 2 describes QED, and Section 3 is devoted to Wee. Then we will present and discuss our results in Section 4.

2 The QED System

2.1 Pre-processing and Indexing

The ACQUAINT document collection which forms the basis for TREC-2004 was pre-processed with a set of Perl scripts, one per newspaper collection, to identify and normalize meta-information. This meta-information included the document ID and paragraph number, the title, publication date and story location. The markup for these last three fields was inconsistent, or even absent, in the various collections, and so collection-specific extraction scripts were required.

The collection was tokenized offline using a combination of the Penn Treebank sed script and Tiphaine Dalmas' Haskell tokenizer. Ratnaparkhi's MXTERMINATOR program was used to perform sentence boundary detection (Reynar and Ratnaparkhi, 1997). The result was indexed with the Managing Gigabytes (MG 1.3g) search engine (Witten et al., 1999). For our TREC-2004 experiments, we used case-sensitive indexing without stopword removal and without stemming.

2.2 Retrieval and Passage Segmentation

Using ranked document retrieval, we obtained the best 100 documents from MG, using the query generated from the question. Since our approach involves full parsing to obtain detailed semantic representations in later stages, we need to reduce the amount of text to be processed to a fraction of each document. To this end, we have implemented QTILE, a simple query-based text segmentation and passage ranking tool. This "tiler" uses the words in the query to extract from the set of documents returned by MG, a set of segments ("tiles"). It does this by shifting a sliding window sentence by sentence over the text stream, retaining all window tiles that contain at least one

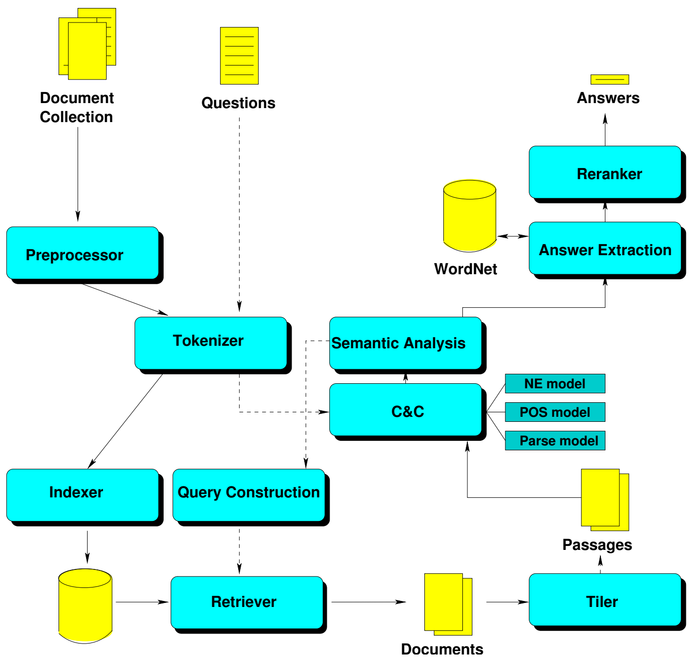


Figure 1: The QED system architecture. (Dashed lines represent processing streams for questions, while solid lines represent processing streams for answers.)

of the words in the query as well as all upper-case query words.

Each tile gets assigned a score based on the following: the number of non-stopword query word tokens (as opposed to types) found in the tile; capitalization agreement between the appearance of a term in the query and its appearance in the tile; and the occurrence of 2-grams and 3-grams in both question and tile. The score for every tile is multiplied with a window function (currently a simple triangle function) which weights sentences in the centre of a window higher than in the periphery.

The tiler is implemented in C++, with linear asymptotic time complexity and constant space requirements. For TREC-2004 we used a window size of 3 sentences and pass forward the top-scoring 100 tiles (with duplicates eliminated using a hash signature test).

2.3 Syntactic and Semantic Analysis

We used the C&C parser to parse the question and the text segments returned by the tiler and Wee. The C&C parser does POS-tagging (Curran and Clark, 2003a) and named entity recognition (Curran and Clark, 2003b), identifying named entities from the standard MUC-7 data set (locations, organisations, persons, dates, times and monetary amounts) and then returns CCG derivations, which are mapped into semantic representations (Bos et al., 2004). This linguistic analysis is applied both to the question under consideration and the text passages that might contain an answer to the question. The semantic analysis forms the basis for query generation, which is basically a list of the lemmas of the content expressions.

Our semantic formalism is based on Discourse Representation Theory (Kamp and Reyle, 1993), but we use extended Discourse Representation Structure (DRS), combining semantic information with syntactic and sortal information. DRSs are defined as ordered pairs of a set of discourse referents and a set of DRS-conditions. The following types of basic DRS-conditions are considered: `pred(x,S)`, `card(x,S)`, `event(e,S)`, and `argN(e,x)`, `rel(x,y,S)`, where `e`, `x`, `y` are discourse referents, `S` a constant, and `N` a number between 1 and 3. Questions introduce a special DRS-condition of the form `answer(x,T)` for a question type `T`. We call this the *answer literal*; answer literals play an important role in answer extraction.

Implemented in Prolog, we reached a semantic coverage of around 95%. Each passage or question is translated into one single DRS; hence DRSs can span several sentences. To deal with pronouns in the questions, basic techniques for pronoun resolution are implemented as well. A set of DRS normalisation rules are applied in a post-processing step, thereby dealing with active-passive alternations, question typing, inferred semantic information, and the disambiguating of noun-noun compounds. The resulting DRS is enriched with information about the original surface word-forms and POS-tags (see Figure 2).

```
id(['QID':36.3,
    'TRECTYPE':
    'FACTOID'],1).

%%% Khmer Rouge
%%% Who was its first leader ?

sem(1,

  [p(1001,'Khmer'),
   p(1002,'Rouge'),
   p(2001,'Who'),
   p(2002,was),
   p(2003,its),
   p(2004,first),
   p(2005,leader),
   p(2006,?)],

  [i(1001,'NNP'),
   i(1002,'NNP'),
   i(2001,'WP'),
   i(2002,'VBD'),
   i(2003,'PRP$'),
   i(2004,'JJ'),
   i(2005,'NN'),
   i(2006,'.')],

  [drs([1000:x0,
        2001:x2,
        2002:e3,
        2003:x1],
       [1000:pred(x0,topic),
        1001:ne(x0,'I-PER'),
        1001:pred(x0,'Khmer'),
        1002:ne(x0,'I-PER'),
        1002:pred(x0,'Rouge'),
        1002:pred(x0,single),
        2001:answer(x2,general,person),
        2002:arg1(e3,x2),
        2002:arg2(e3,x1),
        2002:event(e3,be),
        2003:pred(x0,neuter),
        2003:pred(x0,single),
        2003:rel(x1,x0,of),
        2004:pred(x1,first),
        2005:pred(x1,leader),
        2005:pred(x1,single) ])]

 ).
```

Figure 2: Example of an extended DRS for TREC-2004 question 36.3. The words and POS-tags are co-indexed with the discourse referents and DRS-conditions, and the DRS is enriched with information produced by the named entity recogniser.

2.4 Question Analysis: Evaluation

The C&C parser used in the system has been trained on the CCG version of the WSJ Penn Treebank Wall Street Journal Corpus. The original parser performs extremely poorly on questions, due to the small number of questions in the Treebank. However, in Clark et al. (2004) we show how the parser can be rapidly ported to the question domain.

The novel porting method relies on the separation of the CCG parsing task into two subtasks: *supertagging*, in which CCG lexical categories are assigned to words, and then a final parsing phase in which the lexical categories are combined together, producing a parse tree. Since lexical categories contain so much syntactic information, supertagging can be thought of as *almost parsing*, to borrow a phrase from the TAG parsing literature (Bangalore and Joshi, 1994).

Clark et al. (2004) show that, by marking up new data at the lexical category level only, and using a newly trained supertagger with the original parsing model, high accuracy can be achieved for parsing questions. The advantage of this method is that marking up lexical category data is easier than marking up full derivation trees.

In order to adapt the supertagger to questions, we took around 1,500 questions from the TREC competitions for the years 2000–2003. The questions were automatically POS-tagged and then annotated with lexical categories by Clark, who also corrected any errors made by the POS tagger. The creation of the new question corpus took only a few weeks. The supertagger was then retrained on this new question data. The combination of the new supertagger with the original parsing model is sufficient to produce a highly accurate parser of questions.

This is shown by the parser's performance on the TREC-2004 questions. Of the 286 factoid and list questions, the parser produced 277 analyses, yielding a semantic coverage of 97%. The number of reasonably correct question-DRSs produced for these analyses was 252 (88% of the total). Incorrect analyses were due to tokenisation problems, POS-tagging errors, CCG-categories that did not appear in the training set, and pronoun resolution. Of the 143 cases of pronouns appearing in the questions, 127 (89%) were correctly resolved. The others were resolved incorrectly due to number disagreement of target and pronoun.

2.5 Answer Extraction

The answer extraction component takes as input a DRS for the question, and a set of DRSs for selected passages. It extracts answer candidates from the passages by matching the question-DRS and a passage-DRS, using a relaxed unification method and a scoring mechanism indicating how well the DRSs match each other.

Matching takes advantage of Prolog unification, using Prolog variables for all discourse referents in the question-DRSs, and Prolog atoms in passage-DRSs. It attempts to unify all terms of the question-DRSs with terms in a passage-DRS, using an A* search algorithm. Each potential answer is associated with a score, which we call the DRS score. High scores are obtained for perfect matches (i.e., standard unification) between terms of the question and passage, low scores for less perfect matches (i.e., obtained by "relaxed" unification). Less perfect matches are granted for different semantic types, predicates with different argument order, or terms with symbols that are semantically familiar according to WordNet (Fellbaum, 1998).

After a successful match, the answer literal is identified with a particular discourse referent in the passage-DRS. This is possible because the DRS-conditions and discourse referents are co-indexed with the surface word-forms of the source passage text (see Figure 2). This information is used to generate an answer string, simply by collecting the words that belong to DRS-conditions with discourse referents denoting the answer. Finally, all answer candidates are output in an ordered list. Duplicate answers are eliminated, but answer frequency information is added to each answer in this final list.

Figure 3 shows an example output file. The columns designate the question-id, the source, the ranking score, the DRS score, the frequency of the answer, and a list of sequences of surface word-form, lemma, POS-tag and word index. The best answer is selected from this file by calculating a weighted score of the DRS score and frequency. The weights differ per question type, and were determined by running experiments over the TREC-2003 data.

3 The Wee System

3.1 Overall Strategy

Wee is a web based Question Answering system interfacing an information fusion module, QAAM (Question Answering Answer Model). QAAM is based on the Model-View-Controller design pattern which states that data processing and data rendering should be properly distinguished when engineering a system that deals with both. We apply this pattern to Question Answering: results found on the web are merged into a model and various controllers can access this model and propose a view to the end user. A model may contain several answers at different levels of granularity or aggregation, as well as alternative answers. It may also contain background information, i.e. information that does not correspond to a direct answer but may help for further interpretation.

A QAAM model is a graph where nodes represent concepts and edges express relationships between them. For instance, for the infamous question *Where is the Taj Mahal?*, a QAAM model may contain the following nodes: {*Agra, India, history, women, Atlantic City, New Jersey, casino, resort*}. We discuss below what relations are used and how they are inferred.

Once a model is generated, a controller can query it and render the provided information. We have a special renderer for TREC, but other renderers can be developed as well: full text renderer, summarizer, multi-media renderer (that grabs pictures on the web related to the information contained by the model), more interesting: a dialog controller that goes back and forth between the user and the model, and can eventually enrich the current model by launching new QA processes.

The Wee/QAAM architecture consists of three parts: (1) question analysis and web retrieval, (2) model generation and (3) rendering. The final rendering consists in finding a supporting document and collecting answers for one target.

3.2 Question Analysis

Linguistic processing Before being tokenized, each question is reformulated using the target and the question type provided by TREC, by inserting the target as a topicalised expression at the end of the question. This technique allows us to introduce the target into the question without performing pronoun resolution. In the worst case, the question contains redundant information which will in any case be filtered by the query generation module. This was done for both factoid and list questions. For 'other' questions, the reformulated question is simply the target, understood by default as a definition question.

The question is then tokenized and POS-tagged. We tried out two POS-taggers (C&C and Lingua POS-TAG). It is known that standard POS-taggers do not perform well on questions (for instance, they tend to mistag auxiliary verbs). Therefore, we chose to split the POS-tagging into two steps: using an off-the-shelf tagger followed by a supertagger that we developed ourselves. The supertagger takes as a parameter the tagset of the first tagger and map it onto a smaller set of supertags. This mapping takes into account standard errors from POS taggers and corrects them. Finally, tokens are transformed to lower case except tokens corresponding to named entities, and grouped by their tag family. Also, quotes are preserved and verbs inflected.

Question typing Wee performs question typing based on five features: the wh-type, the wh-complement, the lexical head of the first NP and lexical head of the auxiliary verb group, the modifier of the first NP (if there is one), and the list of the remaining NPs. This is basically done by a series of look-up processes, which terminate when a question is fully disambiguated. (If this process

```
1394 NYT19990821.0176 0.0687983 0.50 8 Degnan Degnan NNP 157001
1394 NYT19990821.0176 0.0687983 0.43 3 the the DT 158010 nation nation NN 158011
1394 APW19990616.0182 0.0923594 0.37 1 Tarzan Tarzan NNP 21011
1394 APW20000827.0133 0.0651768 0.37 2 English English NN 219015
1394 APW20000827.0133 0.0651768 0.37 1 Additionally Additionally NNP 220001
1394 APW20000827.0133 0.0651768 0.37 4 the the DT 220010 U.S. U.S. NNP 220011
```

Figure 3: Example output file of answer extraction.

```
temperature      time
currency         any_time
monetary_value   composition
percentage       effect
weight           purpose
distance         explanation
duration         famous_for
frequency        quote
size             title
age              formula
speed            hyponym
numeric          translation
quantity         acronym
code             term
spatial          definition
any_spatial      proper_name
```

Figure 4: Question types used in Wee

fails, `definition` will be selected as question type). The question types used in Wee are listed in Figure 4.

We distinguish two types of location and time questions. *Any_time* corresponds to a simple *when* question which requires some query expansion. *Time* means there is already a lexical item indicating a time unit, such as in *What year was X* or *When is X's birthday*. The same distinction is made for location questions (*where* versus *what country*).

Web query generation The query sent to the web combines the question phrases and expansion keywords selected according to the question type. We used Google as a web search engine, exploiting its special operators, for instance *i..j* to search on numbers (e.g. *1900..2000* searches for all numbers between 1900 and 2000). We did not use Google's *define* operator, as it often leads to no results, either because no dictionary has the requested entry or because the word is rare or spelled incorrectly. But we used the number range operator for all the questions expecting a numeric answer.

The IR process consists in a relaxation loop that starts with a first query that is highly specific and is relaxed if too few answers have been found. The first query is generated by quoting all the NP and verb expressions and combining them with a first series of expansion keywords. This query can then be relaxed by breaking it down into tokens. The second query also uses a different set of expansion keywords (usually fewer).

Web Filtering and Reranking Once a query has been generated, we simply ask Google for 100 snippets, which are then split into sentences and tokenized. To rerank sentences, we use a scoring based on pattern matching, question words count and the number of different potential answer words.

A penalty filter is also used to remove web noise, notably "sponsored links" and snippets such as *the 1989 World Book Dictionary definition of* which indicate a good document but are not contentful as snippets. Snippets coming from certain websites are also penalized, for instance *trec.nist.gov* and the Answer Bus web sites. Those contain typical QA keywords that add noise to our process (although for our first internal evaluation on TREC 10, they provided many good answers).

Each sentence receives a score according to the following four criteria:

1. *Minimize the penalty score*
   The penalty score is computed on the basis of the penalty filter described above and the number of different potential answer words (i.e. words that are not question or stopwords). If there is no potential answer word, the sentence is highly penalized; otherwise it gets as many points as different answer words.

2. *Maximize the question word percentage*
   If there are repeated question words, the sentence is penalized; otherwise it gets a score between 0.1 and 1 indicating the percentage of question words that have been found. If there are no question words the score is nonetheless slightly raised (to 0.1) because it is still better than a sentence with too many question words. (We call those "spam snippets".)

3. *Maximize the 'be' score*
   A sentence containing the inflected verb *be* (e.g. *was, were, is, are*) indicates a potentially useful syntactic structure and therefore a good basis for answer extraction. Sentences from snippets are actually more often phrases or unfinished sentences.

Therefore this is a good indicator of richer information.

4. *Maximize the clue score*
   To each question type is associated a list of answer clues defined as regular expressions. Those are patterns for clues, not answers. For instance, `currency` is a clue to find a currency name in snippet sentences but it is not an answer. So far we have overall 183 patterns for clues that have been gathered manually.

Sentences for which the penalty score is too high or that do no have any clue are simply removed from the candidate set. 100 web snippets usually generate around 300 sentences. After filtering, we only have around 100 sentences left. Those sentences are then passed on to the modeling module.

3.3 Modeling

As mentioned above, Wee passes its output to QAAM to generate a model based on a graph structure. Modeling consists of two steps: (1) *projection*, the process of mapping a set of sentences to a set of nodes; and (2) *linking*, the process of discovering relationships between the nodes.

Projection The generation of relevant nodes is done by passing pairs of sentences through the Longest Common Substring (LCS) dynamic programming matrix. Wee is implemented in Haskell and makes use of a lazy algorithm to avoid computing a complete matrix when not necessary. This LCS algorithm was adapted to our needs: instead of comparing the pair of sentences character by character, we compare them token by token using a fuzzy match function based on a lazy version of the edit distance algorithm. For efficiency, each match is cached.

To see this, consider the subset of sentences produced from a Google query for the question *What diseases are prions associated with?* as shown in Figure 5. First a cache is computed comparing normalized tokens with the edit distance algorithm. The acceptable distance is dynamic, depending on the length of the strings compared. In our example, *Encephalopathies, ENCEPHALOPATHIES* and *Encephalopathy* are considered equal. The LCS is then computed for each pair of sentences. For our example we get as matching sublist of tokens:

$$\{\{\textit{Spongiform, Encephalopathies}\},\\ \{\textit{SPONGIFORM ENCEPHALOPATHIES}\},\\ \{\textit{Spongiform, Encephalopathy}\}\}.$$

We use the edit distance rather than NLP techniques such as lemmatization or stemming because it allows a match not only between words having the same root but also between words that have been misspelled, which is quite frequent in data coming from the web.

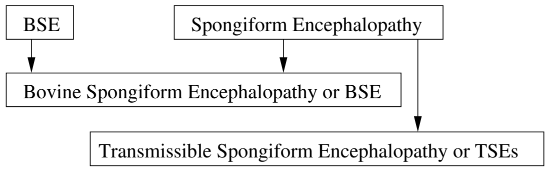
Next, substrings are trimmed of stop words and question words if required. For instance, a match such as {*diseases, called, Transmissible, Spongiform, Encephalopathies*} is trimmed to {*Transmissible, Spongiform, Encephalopathies*} because *diseases* is a question word and *called* is considered a stop word. Substrings are then checked against answer patterns (regular expressions manually gathered for each question type) and answer stop words, i.e. web stop words or usual stop words. Phrases left after this filtering are considered as good candidates and are selected to be model nodes.

Linking For each pair of nodes, we select the relations that characterize them. In the current implementation of QAAM we use three relations: two nodes can be `EQUIVALENT`, one can `OCCURS_IN` the other (and vice-versa with `HAS_OCCURRENCE`) or they can be `DIFFERENT`. These relations are inferred by computing the intersection of content words between phrases. For instance, we have:

$$\{\textit{Spongiform, Encephalopathy}\}\\ \texttt{OCCURS\_IN}\\ \{\textit{Transmissible, Spongiform, Encephalopathies}\}$$

which gives us a simple notion of entailment, namely *Transmissible Spongiform Encephalopathies* → *Spongiform Encephalopathy*).

This is all based on string matching and thus very shallow. Nevertheless, we end up with graph partitions focusing on specific topics, such as:


BSE
Spongiform Encephalopathy
Bovine Spongiform Encephalopathy or BSE
Transmissible Spongiform Encephalopathy or TSEs

This technique has the advantage of getting a better frequency count by taking into account co-occurences of words. We also distinguish between families of answers, which prevents the system from producing redundant answers. This selection of graph partitions is actually part of the role of the controller in charge of the final rendering.

3.4 Rendering

Once a model has been generated, one can choose an appropriate controller to provide one or more answers. It is actually possible to select the whole graph, which would correspond to a detailed answer. Alternatively, one can simply select one representative per family (or partition) to build a more compact answer.

http://www.portfolio.mvm.ed.ac.uk/	Transmissible Spongiform Encephalopathies .
http://kobiljak.msu.edu/CAI/Pathology/	SPONGIFORM ENCEPHALOPATHIES (PRION DISEA SES) A. INTRODUCTORY CONCEPTS ; CHARACTERISTICS OF PRIONS 1.
http://www.bseinfo.org/dsp/dsp	locationContent BSEInfo.org The Source For Bovine Spongiform Encephalopathy ...

Figure 5: Google snippets for question 10.3 *What diseases are prions associated with?*

Interesting nodes are usually specific nodes, i.e. those that do not have children generated by `OCCUR_IN` relations. For instance, within the encephalopathy family, {*Transmissible, Spongiform, Encephalopathies, or, TSEs*} is a good candidate because many nodes in the family have occurences in this node, but it does not occur itself as a whole in another node.

However, for the TREC exercise we chose to output all the nodes, in order to link the answer back to the AQUAINT corpus, a task for which reformulations of the same type of answer are useful: an answer that is too specific can be difficult to find in the TREC corpus. The controller we used outputs each node of the graph ordered by their partition size. Members of a large family are output first, giving preference to the most specific nodes.

3.5 Selecting a supportive TREC document

Once we have found a web answer with Google, we still need to find a supporting document in the AQUAINT corpus collection in order to meet the requirements for the TREC evaluation exercise. For this we implemented a TREC controller, using the Lucene search engine on the AQUAINT corpus.

The TREC controller outputs a web answer with two Lucene queries. The first query looks for a co-occurrence of the answer words and question words within a window of 100 tokens (in a TREC document). The second query is the relaxed version of the first: it looks for at least the answer words and if the documents also contains some question words, its ranking is boosted. In both queries, the target is a required element.

The second query is only used if the first query was not successful. For each document, we take all the sentences where words of the query co-occur. The sentence that maximize the co-occurrence score is output first.

4 Evaluation and Discussion

4.1 Experimental Setup

Three runs were submitted. Run A (Edin2004A) was solely produced by the Wee system, producing web answers for which a supporting document had been found. Run B (Edin2004B) was produced by the QED system using text extractions produced by Wee. Run C (Edin2004C), finally, was produced by the QED system using traditional IR methods plus text extractions produced by Wee. In both runs B and C, the answer produced by Wee was proposed if QED was not able to find one. We expected Run A to perform fairly well (based on judgements on TREC 2002 questions), and Run B to have more exact answers then Run A (which lacks a sophisticated linguistic analysis). We feared answers found by Run A and B not to be supported by the ACQUAINT corpus, and hoped that Run C would score better on this aspect of evaluation.

4.2 Results

Factoid questions formed the majority of the questions at the TREC 2004 QA evaluation exercise. Our results over 230 factoid questions are listed in the table below, where W is the number of wrong, U the number of unsupported, X, the number of inexact and R the number of correct answers.

Run	W	U	X	R	Accuracy
A	166	18	25	21	0.091
B	167	14	16	33	0.143
C	194	7	8	21	0.091

As expected, the number of inexact answers was high for runs A and B. (Closer inspection of our inexact answers revealed that the judges were very strict this year in assessing inexact answers.) The number of unsupported answers was also substantial for both runs A and B. Run C was slightly disappointing, we were expecting it to get a better performance.

Our 'best' answer was *Saloth Sar* to question 36.3 *Who was its first leader?* in the context of target *Khmer Rouge*. Out of 63 runs we were the only submission that got a correct answer for this question. It is interesting to note that the overlap of our two best runs (A and B) is only 8 correct answers. A better answer selection component could considerably improve our overall system. The table below shows the difference in performance of the three runs distributed over Wee question types, summing over correct, inexact and unsupported answers:

Question Type	Total	Run A	Run B	Run C
any_time	48	27 (54%)	26 (54%)	13 (27%)
proper_name	38	14 (37%)	6 (16%)	4 (10%)
any_spatial	24	6 (16%)	8 (33%)	3 (12%)
hyponym	23	4 (17%)	5 (22%)	2 (09%)
quantity	21	0 (00%)	3 (14%)	3 (21%)
spatial	12	4 (33%)	3 (25%)	1 (08%)
term	11	1 (09%)	0 (00%)	1 (11%)
time	9	1 (11%)	4 (44%)	3 (33%)
definition	8	1 (12%)	2 (25%)	1 (12%)
famous_for	7	1 (14%)	1 (14%)	1 (07%)
title	6	2 (33%)	3 (50%)	3 (50%)
explanation	6	0 (00%)	0 (00%)	0 (00%)
duration	4	0 (00%)	0 (00%)	0 (00%)
monetary_value	3	2 (66%)	2 (66%)	1 (33%)
acronym	3	0 (00%)	0 (00%)	0 (00%)
composition	2	1 (50%)	0 (00%)	0 (00%)
purpose	2	0 (00%)	0 (00%)	0 (00%)
acronym	1	0 (00%)	0 (00%)	0 (00%)
frequency	1	0 (00%)	0 (00%)	0 (00%)
speed	1	0 (00%)	0 (00%)	0 (00%)
Total	230	64 (28%)	63 (27%)	36 (16%)

We didn't devote much of our research time to list questions, and the bad results clearly underline this. For the 55 list questions we got an average F score of 0.036 for run A, 0.054 for run B, and 0.043 for run C. For our analysis on 'other' questions, a similar story can be told, although the results are not as bad as for the list questions. Over the 64 'other' questions we achieved an average F score of 0.068 for run A, 0.152 for run B, and 0.194 for run C. The latter figure is better higher than the medium score of all the 63 submitted runs. The final scores for our three runs were respectively 0.072 (run A), 0.123 (run B), and 0.105 (run C).

4.3 Discussion

Compared to TREC 2003, the two major improvements of the QED system are the use of a more fine-grained question-type ontology, and the utilisation of the CCG parser, accomplishing both higher coverage and precision on both questions and answers. The Wee system, developed by Tiphaine Dalmas, was a completely new component of our TREC-2004 setup.

In TREC 2004, the overall accuracy of factoid questions of the 63 runs submitted to the QA track ranged between 0.009 and 0.770 (median 0.170). For list questions, the best, median, and worst average F-scores were 0.622, 0.094, and 0.000, respectively. For 'other' questions, the F-scores ranged from 0 to 0.460 (with a median of 0.184).

The results of the three runs indicate that using the Web for finding answers rather than using standard IR gives better scores for factoids, but not for definition questions. Deep linguistic processing tools gives more exact answers, although the exactness of answers requires considerable improvement in our current system.

Acknowledgements

Dalmas is supported by the School of Informatics, University of Edinburgh. Leidner is supported by the German Academic Exchange Service (DAAD) under scholarship D/02/01831 and by Linguit GmbH (research contract UK-2002/2).

References

[Bangalore and Joshi1994] Srinivas Bangalore and Aravind Joshi. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th COLING Conference*, pages 154–160, Kyoto, Japan.

[Bos et al.2004] Johan Bos, Stephen Clark, Mark Steedman, James R. Curran, and Julia Hockenmaier. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, Geneva, Switzerland.

[Clark et al.2004] Stephen Clark, Mark Steedman, and James R. Curran. 2004. Object-extraction and question-parsing using ccg. In *Proceedings of the SIGDAT Conference on Empirical Methods in Natural Language Processing (EMNLP'04)*, pages 111–118, Barcelona, Spain.

[Curran and Clark2003a] James R. Curran and Stephen Clark. 2003a. Investigating GIS and smoothing for maximum entropy taggers. In *Proceedings of the 11th Annual Meeting of the European Chapter of the Association for Computational Linguistics (EACL'03)*, pages 91–98, Budapest, Hungary.

[Curran and Clark2003b] James R. Curran and Stephen Clark. 2003b. Language independent NER using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning (CoNLL-03)*, pages 164–167, Edmonton, Canada.

[Fellbaum1998] Christiane Fellbaum, editor. 1998. *WordNet. An Electronic Lexical Database*. The MIT Press.

[Kamp and Reyle1993] Hans Kamp and Uwe Reyle. 1993. *From Discourse to Logic. An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT*. Kluwer, Dordrecht.

[Leidner et al.2004] Jochen L. Leidner, Johan Bos, Tiphaine Dalmas, James R. Curran, Stephen Clark, Colin J. Bannard, Mark Steedman, and Bonnie Webber. 2004. The QED open-domain answer retrieval system for TREC 2003. In *Proceedings of the Twelfth Text Retrieval Conference (TREC 2003)*, NIST Special Publication 500-255, pages 595–599, Gaithersburg, MD.

[Reynar and Ratnaparkhi1997] Jeffrey C. Reynar and Adwait Ratnaparkhi. 1997. A maximum entropy approach to identifying sentence boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, D.C.

[Witten et al.1999] Ian A. Witten, Alistair Moffat, and Timothy C. Bell. 1999. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, Los Altos, CA, 2nd edition.