AnswerFinder in TREC 2003

Diego Mollá
Centre for Language Technology
Macquarie University
Sydney, NSW 2109
Australia
Tel. +61 2 9850 9531
Fax +61 2 9850 9551
`diego@ics.mq.edu.au`

Abstract

In this our first participation in TREC we have focused on the passage task of the question answering track. The main aim of our participation was to test the impact of various types of linguistic information in a simple question answering system. In particular, we have tested various combinations of word overlap, grammatical relations overlap, and overlap of minimal logical forms in the final scoring module of the system. The results indicate a small increase of accuracy with respect to a baseline system based on word overlap. Overall, given the short time available for developing the system, the results are satisfactory and equal or surpass the median.

1 Introduction

This is the first time that the Centre for Language Technology at Macquarie University participates in TREC. Due to strong time restrictions we decided to implement a simple and functional system for the passage task of the Question Answering track. The final estimated time of development was about 55 person-hours (plus execution time) distributed among three months of intermittent work.

Section 2 describes the general architecture of the system. Our main focus was the exploration of sentence similarity measures for question answering. The measures used were based on word overlap, grammatical relations, and minimal logical forms. The two latter measures are described in Sections 3 and 4, respectively. Section 5 presents the final results and discussion. Finally, Section 6 describes a post-submission extension that incorporates question classification and named-entity recognition.

2 System Architecture

The system is fairly straightforward (Figure 1). A **document preselection** stage returns the documents preselected by NIST. A subsequent **sentence preselection** stage splits the documents into sentences and ranks the sentences according to word overlap. A final **scoring** stage analyses the top-ranking sentences returned by the previous stage and re-ranks the sentences according to a similarity measure. The top-ranking sentence is returned as the answer, possibly truncated if it is longer than the limit of 250 characters. Note that all questions are treated the same way. In other words, there is no question classification stage. Also, no attempt to detect NIL answers was made.

It is worth noting that the complete process was done when the question was processed. All the data structures (except, of course, the documents provided by NIST) were built on the fly.

To determine the number of documents to preselect, during the development of the system we analysed the questions used in TREC 2002 and the answers available from the TREC web pages. In particular, we used the regular expressions provided by Ken Litkowsky to determine if an arbitrary document contained the answer of a specific question. The results of this analysis are

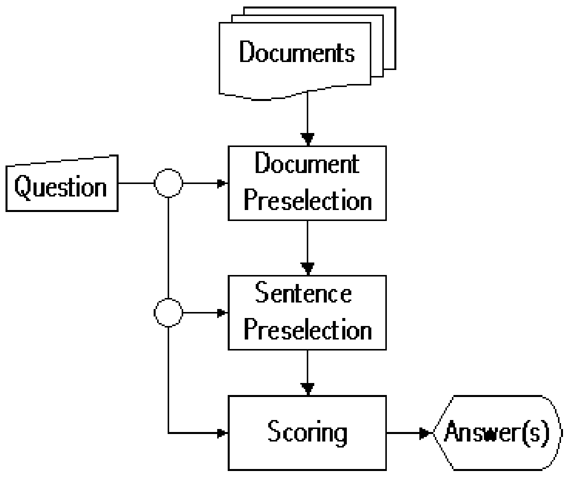

Figure 1: Architecture of AnswerFinder.

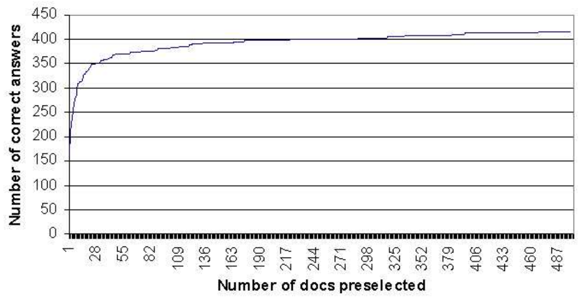

Figure 2: Relation between preselected documents and number of correct answers.

not necessarily accurate for two reasons. First of all, good answers phrased in unfamiliar terms may not be covered by the regular expressions. Second, some text may happen to match a regular expression by coincidence but still the document may fail to support the answer. Still, the results are indicative for our purposes. The results of our analysis is summarised in Figure 2, which shows the number of questions that have an answer within the top X documents. We can see that there is little gain by preselecting many documents. For practical reasons we set the threshold of documents to preselect to $X = 50$. For that threshold, 74% of the questions would find a document that satisfies the regular expression of the answer.

To split the documents into sentences we used a simple regular expression that detects punctuation characters as end-of-sentence markers:

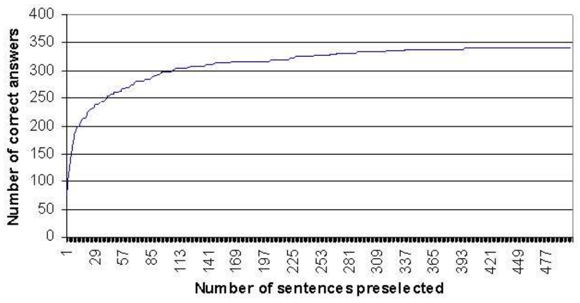

Figure 3: Relation between preselected sentences and number of correct answers (using the top 50 documents).

```
'(?:\.|\?|!|;|<.*?>)+'
```

The resulting sentences are ranked according to word overlap. After several experiments we found that the optimal measure of word overlap is obtained by using a list of stop words[1] and ignoring repeated words in the answer candidate. We arbitrarily set a threshold of 100 sentences to be sent to the final scoring stage. Figure 3 shows the relation between the sentences preselected and the number of correct answers. For the threshold of 100 sentences and using the 2002 question set, 59.4% of the questions would have a string that satisfies the regular expression of the answer. This is therefore the expected upper limit of accuracy that the scoring module can achieve.

The final scoring module combines several types of information. Apart from word overlap we experimented with other types of overlap that use various types of linguistic information. In particular, we used grammatical relations and minimal logical forms, as described in the following sections.

3 Grammatical Relations

The grammatical relations by Carroll et al. (1998) were devised to enable comparative evaluations of parsers. Following their evaluation methodology, the output of the parsers to evaluate is converted into sets of grammatical relations, thus enabling the representation of the parser output in a uniform way. In order

[1]We used the list of stop words available from `http://www-fog.bio.unipd.it/waishelp/stoplist.html`

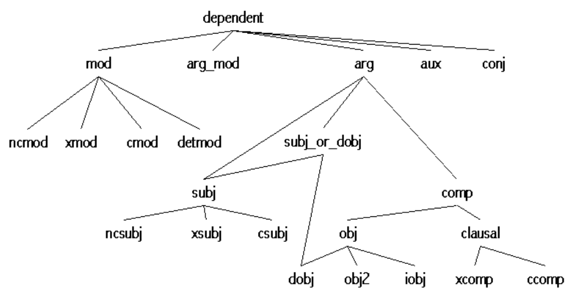

Figure 4: Hierarchy of grammatical relations (Briscoe and Carroll, 2000).

to be able to accommodate parsers with different granularity in the output, the grammatical relations are classified hierarchically (Figure 4). Different types of parser output are represented with different types of grammatical relations and therefore a wide range of parsers can be easily compared.

Table 1 lists the grammatical relations used in the examples of this paper and the final implementation. For further detail about grammatical relations see (Briscoe and Carroll, 2000).

For example, the grammatical relations for the sentence *The man that came ate bananas and apples with a fork without asking* are:

```
DETMOD(_,man,the),
CMOD(that,man,come),
SUBJ(come,man,_),
SUBJ(eat,man,_),
DOBJ(eat,banana,_),
DOBJ(eat,apple,_)
CONJ(and,banana,apple),
NCMOD(fork,eat,with),
DETMOD(_,fork,a),
XCOMP(without,eat,ask)
```

Briscoe and Carroll's grammatical relations are different from the dependency arcs used in dependency grammar formalisms (Mel'čuk, 1988). Consider *The man that came ate bananas and apples with a fork*. Figure 5 (a) shows the graphical representation of the structure returned by Conexor FDG, a dependency-based parsing system (Tapanainen and Järvinen, 1997). For comparison, Figure 5 (b) shows a simplified graphical representation of the grammatical relations. In dependency grammar a unique head is assigned to each word, thus the head of *man* is *ate*. However *man* is the dependent of more than one grammatical relation, namely `SUBJ(eat,man,_)` and `SUBJ(come,man,_)`. Furthermore, in dependency grammar a word can have at most one dependent of each argument type, and so *ate* can have at most one object. But the same is not true for grammatical relations, and we get both `OBJ(eat,banana,_)` and `OBJ(eat,apple,_)`. Thus, grammatical relations can theoretically provide a sentence representation that is closer to the semantic contents of a sentence than the representation provided by dependency arcs. In practice, the representational power of the grammatical relations depends on the output of the parser used.

Grammatical relations can be used to introduce parser-independent syntactic information in a question-answering system. Since the grammatical relations are expressed as *lists of relations*, a score measure can be implemented by simply computing the *overlap* of grammatical relations between the question and the answer candidate. In theory, to compute the overlap we must use the hierarchical organisation of the grammatical relations to decide if two grammatical relations are related. For example, `SUBJ(eat,man,_)` can be subsumed by `SUB_OR_DOBJ(eat,man)`. However, since the same parser was used for both the question and the answer, the granularity of grammatical relations between questions and answer candidates will be practically the same. Thus, each grammatical relation can be seen as an unstructured token and the scoring module can simply count the number of common tokens, very much like counting the overlap of words. This was the approach used in our QA prototype.

4 Minimal Logical Forms

Flat logical forms have been used in several NLP systems, including question-answering systems (Harabagiu et al., 2001; Lin, 2001; Mollá et al., 2000, for example). The flat logical forms that we use in our QA system are borrowed from (Mollá et al., 2000), who uses reification to flatten out nested expressions. For example, the logical form of *The cp command will quickly copy files* is:[2]

[2]For illustration purposes, the logical forms used in this paper are slight variants of the ones shown in the literature.

Relation	*Description*
CONJ(type,head+)	Conjunction
MOD(type,head,dependent)	Modifier
CMOD(type,head,dependent)	Clausal modifier
NCMOD(type,head,dependent)	Non-clausal modifier
DETMOD(type,head,dependent)	Determiner
SUBJ(head,dependent,initial_gr)	Subject
OBJ(head,dependent,initial_gr)	Object
DOBJ(head,dependent,initial_gr)	Direct object
XCOMP(head,dependent)	Clausal complement without an overt subject

Table 1: Grammatical relations used in this paper.

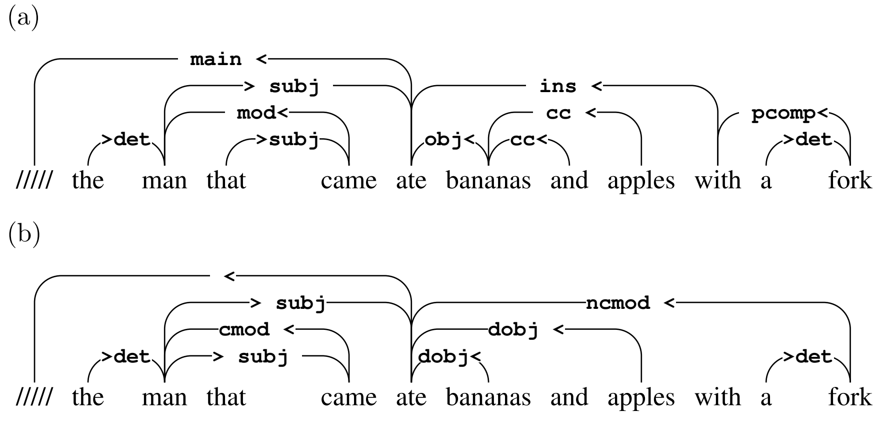
(a)
main <
> subj
ins <
mod<
cc <
pcomp<
>det
>subj
obj<
cc<
>det
///// the man that came ate bananas and apples with a fork
(b)
<
> subj
ncmod <
cmod <
dobj <
>det
> subj
dobj<
>det
///// the man that came ate bananas and apples with a fork

Figure 5: (a) Dependency structure of a sample sentence; (b) grammatical relations.

```
holds(e6),
object('cp',o2,[x2]),
object('command',o3,[x3]),
compound_noun(x2,x3),
prop('quickly',p5,[e6]),
evt('copy',e6,[x3,x7]),
object('file',o7,[x7])
```

The logical form above says that there are two entities x2 and x3 that represent two objects for the compound noun *cp command*. There is an entity x7 (a file); there is an entity e6, which represents a copying event where the first argument is x3 (the object introduced by the head of the compound noun) and the second argument is x7; there is an entity p5 which states that e6 is done quickly, and the event e6 (the copying) holds.

The above expression does not aim to express the complete logical form of the sentence. For example, there is no information about quantification, tense, aspect, and plurals. In essence, only the main relations among open words and determiners is expressed. This is why our logical forms are called *minimal logical forms*: only information that is minimal for the task of question-answering is encoded.

An advantage of the use of flat logical forms over nested logical forms is that, again, sentence similarity can be measured as a type of overlap. The only additional complexity is that the question now contains variables. For example, the minimal logical form of *Which command copies files?* is (the symbols in uppercase indicate variables):

```
object('command',O1,[X1]),
evt('copy',E2,[X1,X2]),
object('file',O2,[X2])
```

If this logical form is to match that of the sentence *The cp command will quickly copy files* above, the scoring module needs to instantiate the variable O1 in the question with the constant o3

Answer candidate	**Minimal Logical Form**
John saw Mary	object('john',o1,[x1]), object('mary',o3,[x3]), evt('see',e2,[x1,x3])

Question	**Minimal Logical Form**
Did John see Mary?	**object('mary',O,[X]), evt('see',E,[Y,X]), object('john',O2,[Y])**
Did Mary see John?	object('john',O,[X]), **evt('see',E,[Y,X])**, object('mary',O2,[Y])

Table 2: Question answering using flat logical forms. Overlap shown in bold.

in the answer candidate, `X1` with `x3`, and so on. In our implementation we have used Prolog unification. Basically, the logical form of the answer candidates is stored as Prolog data, and a simple Prolog program computes the overlap of the logical forms of the answer candidates with the logical form of the question. Also, the scoring module ignores the `holds` term in the question logical form. Otherwise, since almost all questions and candidate answers contain a `holds` term in the logical form, two completely unrelated sentences would have an overlap of 1 and this is counterintuitive.

Since there are several plausible combinations of variable instantiations, the scoring module finds the set of instantiations that provides the highest overlap of logical forms.

Table 2 shows the minimal logical forms of questions that differ solely in the argument positions, the minimal logical form of an answer candidate, and the resulting overlaps.

5 Results and Discussion

To test the impact of grammatical relations and minimal logical forms, we experimented with different scoring modules corresponding to word overlap, grammatical relations, and minimal logical forms, using the TREC 2002 questions. The results (Table 3) show that, remarkably, word overlap is best.

Formula	*Accuracy*
Word overlap	14.8%
Grammatical relations	09.0%
Minimal logical forms	10.8%

Table 3: Experiments with TREC 2002 data.

Due to the limited time available we decided to postpone any cause analysis. Instead we ran several experiments combining the linguistic information available. Table 4 shows the final runs submitted to TREC, the results of our experiments with data from TREC 2003, and the final results returned by NIST.

The data in the 2002 column show that the runs submitted produce slightly better results than simple word overlap. Interestingly, The evaluation provided by NIST (2003 column) gives noticeably better results than our home evaluation with the TREC 2002 data (2002 column). This may be due to the fact that the regular expressions provided by Litkowsky do not attempt to cover all possible formulations of correct answers, or it may be indeed the case that the questions asked in TREC 2003 are easier to process. As we will see in next section, the former is more likely.

6 Adding Named Entities

During the development of the system we tried to integrate the named entity recogniser bundled with GATE.[3]

First of all, we implemented a question analyser module that classifies the question into the type of expected answer. The classifier uses 29 regular expressions to allocate one of the following types to the question: person, date, location, money, number, city, date, organization, location, percent,country, state, river, name, and unknown. The regular expressions were based on the questions used in TREC 2002. The final module has an accuracy of 78.6% (393 from a total of 500 questions were correctly classified).

Since GATE's named entity recogniser can detect a reduced number of named entity types (person, location, date, money, and organization only), a simple mapping was necessary between the question classification and the final list of answer types (Table 5).

[3]`http://gate.ac.uk/`

Run	*Formula*	*2002*	*2003*
answfind1	$3wo + gro$	16.8%	19.1%
answfind2	$9wo + 3gro + mo$	16.8%	18.6%
answfind3	$9wo + 3mo + gro$	15.6%	18.2%

Table 4: Runs submitted to TREC 2003. The 2002 column indicates the results of an automatic self-evaluation with data from TREC 2002. The 2003 column indicates the evaluation results returned by NIST. The formula components are: wo – word overlap; gro – overlap of grammatical relations; mo – overlap of minimal logical forms.

Question Type	*Answer Type*
country, city, state, river	location
percent	number
name	person OR organization OR location
Any other question type yields same answer type	

Table 5: Mapping between question type and answer type.

The information regarding answer type is used during the sentence preselection stage. Thus, the score given to a sentence is the sum of word overlap with the question (as described above) plus a reward of 10 points if the sentence contains an entity of the expected answer type.

We developed a Java interface to GATE's named entity recogniser. However, the steep learning curve required to learn Java, together with unexpected execution errors prevented us from integrating the NE module in the final version. The final system therefore did not use the named entity recogniser and as a consequence the question classifier became useless and therefore it was disabled. Subsequent work on the Java interface enabled us to compute the named entities of the top 50 documents preselected for the TREC 2002 and TREC 2003 questions. With these named entities computed off-line we ran AnswerFinder and obtained the results shown in Table 6.

2002	*Without NEs*	*With NEs*
answfind1	16.8%	19.1%
answfind2	16.8%	19.3%
answfind3	15.6%	18.4%
2003	*Without NEs*	*With NEs*
answfind1	18.2%	16.2%
answfind2	17.4%	15.7%
answfind3	17.2%	15.5%

Table 6: Results of integrating the named entity recogniser.

The results show a noticeable improvement of accuracy in the runs with the TREC 2002 question set. In contrast, accuracy *decreases* in the runs with the TREC 2003 question set. A plausible explanation to the results with the TREC 2003 question set is that we used the new regular expressions provided by Litkowsky for the evaluation. The regular expressions are based on the set of answers returned by the systems competing in TREC 2003. Possibly, some of the answers returned by the version with named entities are paraphrases of the correct answer that do not match the regular expressions and therefore they are erroneously classified as wrong. In fact, note that the results without named entities reported in Table 6 are slightly worse than the ones returned by NIST (Table 4) due to inaccuracies in the regular expressions.

7 Conclusions and Further Research

The short time available only allowed us to build a baseline system for the passages task of the Question Answering track. Still, we were pleased to find that the results were better or equal than the median of all 21 submissions to the task. Overall, our experiments suggest that simple word overlap gives better performance than simple overlaps based on grammatical relations or minimal logical forms alone. These findings confirm the work by (Mollá, 2003), who used a similar question answering system for the Reading Comprehension corpus (Hirschman et al., 1999).

Further work will include:

Error analysis. This will be the first step to do. We will determine if different types of questions are more or less likely to produce good results with the different overlap measures and identify methods of combining word overlap, grammatical relations, and minimal logical forms for each type of question.

NE integration. We will finalise the integration of the named entity recogniser and identify entity types that are useful for the task of question answering. Besides using the named entities to determine the answer candidates, we will also explore ways to include NE information in the parsing modules and semantic interpreter. This way we hope to obtain more accurate grammatical relations and logical forms.

Logical forms. We will explore ways to leverage the use of logical forms by using more sophisticated measures. For example, we will look into adding weights to the logical forms. We will also explore the possibility of using weighted abduction methods.

Extract the exact answer. Logical forms may be useful to determine the exact answer of the question. For example, the original ExtrAns system (Mollá et al., 2000) generates a predicate of the form `object(_,_,_)` that represents the object asked about by the question word. ExtrAns also keeps track of what words produces what predicates in the minimal logical form. All this information can be used to determine the exact part of the sentence that matches the concept being asked about.

Complex questions. We will also work on methodologies to answer questions that require the fusion of output from several documents, such as list and definition questions.

By introducing the above and other extensions, in future participations in the question answering track we hope to increase the accuracy of the system and to participate in the main task of the track.

References

Ted Briscoe and John Carroll. 2000. Grammatical relation annotation. On-line document. `http://www.cogs.susx.ac.uk/lab/nlp/carroll/grdescription/index.html`.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proc. LREC98*.

Sanda Harabagiu, Dan Moldovan, Marius Paşca, Mihai Surdeanu, Rada Mihalcea, Roxana Gîrju, Vasile Rus, Finley Lăcătuşu, and Răzvan Bunescu. 2001. Answering complex, list and context questions with LCC's question-answering server. In Ellen M. Voorhees and Donna K. Harman, editors, *Proc. TREC-10*, number 500-250 in NIST Special Publication. NIST.

Lynette Hirschman, Marc Light, Eric Breck, and John D. Burger. 1999. Deep Read: A reading comprehension system. In *Proc. ACL'99*. University of Maryland.

Jimmy J. Lin. 2001. Indexing and retrieving natural language using ternary expressions. Master's thesis, MIT.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Diego Mollá, Rolf Schwitter, Michael Hess, and Rachel Fournier. 2000. Extrans, an answer extraction system. *Traitment Automatique des Langues*, 41(2):495–522.

Diego Mollá. 2003. Towards semantic-based overlap measures for question answering. In *Proc. ALTW 2003*, Melbourne, Australia.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. In *Proc. ANLP-97*. ACL.