Macquarie University at DUC 2006: Question Answering for Summarisation

Diego Mollá and **Stephen Wan**
Centre for Language Technology
Division of Information and Communication Sciences
Macquarie University
Sydney, Australia
{diego,swan}@ics.mq.edu.au

Abstract

We present an approach to summarisation based on the use of a question answering system to select the most relevant sentences. We used AnswerFinder, a question answering system that is being developed at Macquarie University. The sentences returned by AnswerFinder are further re-ranked and collated to produce the final summary. This system will serve as a baseline upon which we intend to develop methods more specific to the task of question-driven summarisation.

1 Introduction

This document describes Macquarie University's effort to use the output of a question answering system for the task of document summarisation.[1] Current question answering systems in general, and our system in particular, are designed to handle simple questions that ask for a specific item of information. In contrast, a DUC topic is typically a block of text made up of a sequence of declarative sentences. A complex information request such as those formulated as DUC topics needs to be mapped into a set of simpler requests for specific information. This is what we did in our implementation.

We used AnswerFinder, the question answering system being developed at Macquarie University. The main feature of AnswerFinder is the use of lexical, syntactic and semantic information to find the sentences that are most likely to answer a user's question. We decided to make minimal modifications to AnswerFinder and to focus on the conversion of the DUC text into questions that can be handled by AnswerFinder. In particular, the text of each DUC topic was split into individual sentences, and each sentence was passed to AnswerFinder independently. A selection of the resulting sentences were combined using a simple method described here.

Section 2 describes how the DUC topics were converted into individual questions. Section 3 describes the question answering technology used to find the sentences containing the answers. Section 4 focuses on the method used to select the most relevant sentences and combine them to form the final summaries. Sections 5 presents the results. Finally, Section 6 presents some conclusions and discusses further research.

2 Converting Topics to Sequences of Questions

AnswerFinder is designed to answer simple questions about facts and lists of facts. The version of AnswerFinder used here is a Python-based version that participated in the question answering track of TREC 2004 (Mollá and Gardiner, 2005).

Questions in the QA track of TREC 2004 were grouped into topics. Each topic is a group of questions about specific aspects of the topic. An example of a TREC topic is shown in Table 1.

Questions in a topic are of three types:

[1] This work is supported by the Australian Research Council under ARC Discovery grant DP0450750.

Topic number 2: Fred Durst		
2.1	FACTOID	What is the name of Durst's group?
2.2	FACTOID	What record company is he with?
2.3	LIST	What are titles of the group's releases?
2.4	FACTOID	Where was Durst born?
2.5	OTHER	Other

Table 1: A topic in TREC 2004

FACTOID: Questions about specific facts. The answer is a short string, typically a name or a number.

LIST: Questions about lists of facts.

OTHER: This special question requires finding all relevant facts about the topic that have not been mentioned in the answers to the previous questions.

The grouping of TREC questions into topics was very convenient for our purposes, so we decided to convert DUC topics directly into TREC topics. This was done by using the same topic names as in the DUC topics, and splitting the DUC topic descriptions (<narr> field) into individual sentences. Each individual sentence was treated as a LIST question by AnswerFinder. The rationale for converting all sentences into LIST questions is that we aim at finding all sentences that contain the answer. AnswerFinder is designed so that the output of a LIST question is the list of unique answers to that question. This is therefore the closest to what we need.

For example, the DUC topic D0602B (Table 2) was converted into the TREC-like topic shown in Table 3.

The conversion is clearly a simplification of what could be done. There are two enhancements that would be likely to produce questions that are better tuned for AnswerFinder:

- Transform indicative forms into questions. This option seems obvious, though the impact of doing this is not as strong as it would seem. This is so because the question answering module can handle sentences in the declarative form. As a matter of fact, the parser used by AnswerFinder is generally more accurate with declarative sentences than with interrogative sentences. The only real impact of using the interrogative form is during the process of classifying the question. However, given that the kinds of questions asked in DUC are likely to be very different from those asked in TREC, it is still necessary to adapt the question classifier to the new task.

- Split complex sentences into individual questions. For example, the sentence *Include information regarding trends, side effects and consequences of such use* could be converted into three questions:

  1. *What are the trends of such use?*
  2. *What are the side effects of such use?*
  3. *What are the consequences of such use?*

num	D0602B
title	steroid use among female athletes
narr	Discuss the prevalence of steroid use among female athletes over the years. Include information regarding trends, side effects and consequences of such use.

Table 2: DUC topic D0602B

Topic number D0602B: steroid use among female athletes		
D0602B.1	LIST	Discuss the prevalence of steroid use among female athletes over the years.
D0602B.2	LIST	Include information regarding trends, side effects and consequences of such use.

Table 3: DUC topic converted into a TREC topic

3 Sentence Selection Driven by the Questions

The version of AnswerFinder that participated in TREC 2004 returns exact answers. For example,

the exact answer to the question *What is the name of Durst's group?* is *Limp Bizkit*. In the process of finding the exact answer, the system first determines the sentences that are most likely to contain the answer. This is the output that we made AnswerFinder produce for the DUC system. In particular, the following modules of AnswerFinder were used:

- Question normalisation
- Question classification
- Candidate sentence extraction
- Sentence re-scoring

These modules are described below.

3.1 Question Normalisation

Given that the questions may contain anaphoric references to information external to the questions, AnswerFinder performs simple anaphora resolution on question strings. In particular, AnswerFinder performs a simple replacement of the pronouns in the question with the topic text. Since AnswerFinder needs syntactically correct questions, the target is transformed to the plural form or the possessive form where necessary. The transformation uses very simple morphological rules to transform the questions as shown in Table 4.

3.2 Question Classification

Generally, particular questions signal particular named entity types expected as responses. Thus, the example below expects a person's name in response to the question:

> *Who founded the Black Panthers organization?*

AnswerFinder uses a set of 29 regular expressions to determine the expected named entity type. These regular expressions are the same used in our contribution to TREC 2003 and they target the occurrence of *Wh-* question words. In addition, specific keywords in the questions indicate expected answer types as shown in Table 5.

3.3 Candidate Sentence Extraction

Given the set of documents provided by NIST, AnswerFinder selects 100 sentences from these documents as candidate answer sentences.

Candidate sentences are selected in the following way:

1. The documents provided by NIST are split into sentences. The sentence splitter follows a simple approach based on the use of a fixed list of sentence delimiters. This is the same method that we used to split the DUC topic description into TREC questions.

2. Each sentence is assigned a numeric score: 1 point for each non-stopword that appears in the question string, and 10 points for the presence of a named entity of the expected answer type. AnswerFinder automatically tags the named entities in all the documents in an off-line stage prior to the processing of DUC topics.

3. For each question, the 100 top scoring sentences are returned as candidate answer sentences.

As an example of the scoring mechanism, consider this pair of question and candidate answer sentence:

> **Q:** *How far is it from **Mars** to **Earth**?*
>
> **A:** According to evidence from the SNC meteorite, which fell from **Mars** to **Earth** in ancient times, the water concentration in Martian mantle is estimated to be **40 ppm**, far less than the terrestrial equivalents.

The question and sentence have two shared non-stopwords: *Mars* and *Earth*. Further, this sentence has a named entity of the required type (Number): *40 ppm*, making the total score for this sentence 12 points.

3.4 Sentence Re-scoring

The 100 candidate sentences are re-scored based on a combination of lexical, syntactic, and semantic features:

*What record company is **he** with?*	⟶	*What record company is **Fred Durst** with?*
*How many of **its** members committed suicide?*	⟶	*How many of **Heaven's Gate's** members committed suicide?*
*In what countries are **they** found?*	⟶	*In what countries are **agoutis** found?*

Table 4: Examples of pronoun resolution performed by AnswerFinder

Keyword	Answer Type
city, cities	Location
percentage	Number
first name, middle name	Person
range	Number
rate	Number
river, rivers	Location
what is, what are, what do	Person, Organisation or Location
how far, how long	Number

Table 5: Examples of question keywords and associated answer types

Lexical: The presence of a named entity of the expected answer type and the overlap of words.

Syntactic: The overlap of grammatical relations.

Semantic: The overlap of flat logical forms extended with patterns.

The use of lexical information has been described in Section 3.3. Below we will briefly describe the use of syntactic and semantic information.

3.4.1 Grammatical Relation Overlap Score

The grammatical relations devised by Carroll et al. (1998) encode the syntactic information in questions and candidate answer sentences. We decided to use grammatical relations and not parse trees or dependency structures for two reasons:

1. Unlike parse trees, and like dependency structures, grammatical relations are easily incorporated into an overlap-based similarity measure.

2. Parse trees and dependency structures are dependent on the grammar formalism used. In contrast, Carroll et al. (1998)'s grammatical relations are independent of the grammar formalism or the actual parser used. Our choice of parser was the Connexor Dependency Functional Grammar and parser (Tapanainen and Järvinen, 1997). Being dependency-based, the transformation to grammatical relations is relatively straightforward.

An example of the grammatical relations for a question and a candidate answer sentence follows:

Q: *How far is it from Mars to Earth?*
(subj be it _)
(xcomp from be mars)
(ncmod _ be far)
(ncmod _ far how)
(ncmod earth from to)

A: *It is 416 million miles from Mars to Earth.*
(ncmod earth from to)
(subj be it _)
(ncmod from be mars)
(xcomp _ be mile)
(ncmod _ million 416)
(ncmod _ mile million)

The similarity-based score is the number of relations shared between question and answer sentence. In the above example, the two overlapping grammatical relations are shown in **boldface**.

3.4.2 Flat Logical Form Patterns

Semantic information is represented by means of flat logical forms (Mollá, 2001). These logical forms use reification to flatten out nested expressions in a way similar to other QA systems (Harabagiu et al., 2001; Lin, 2001, for example). The logical forms are produced by means of a process of bottom-up traversal of the dependency structures returned by Connexor (Mollá and Hutchinson, 2002).

A straightforward way of using the flat logical forms is to compute their overlap in the same way as we find the overlap of grammatical relations, so that the score of the above example would be computed as follows:

Q: *What is the population of Iceland?*
object(iceland, o6, [x6])
object(population, o4, [x1])
object(what, o1, [x1])
prop(of, p5, [x1, x6])

A: *Iceland has a population of 270000*
dep(270000, d6, [x6])
object(population,o4,[x4])
object(iceland,o1,[x1])
evt(have,e2,[x1,x4])
prop(of,p5,[x4,x6])

In this example, the two overlapping logical form predicates are shown in boldface. Note that the process to compute the overlap of logical forms must map the variables from the question to variables from the candidate answer sentence. AnswerFinder uses Prolog unification for this process.

With the goal of taking into consideration the differences between a question and the various forms that can be used to answer it, AnswerFinder uses patterns to capture the expected form of the answer sentence and locate the exact answer. Thus, if a question is of the form *what is X of Y?*, then a likely answer can be found in sentences like *Y has a X of ANSWER*. In contrast with other approaches, AnswerFinder uses flat logical form patterns. For example, the pattern for *what is X of Y?* is:

Question Pattern:
object(ObjX,VobjX,[VeX]),
object(what,_,[VeWHAT]),
object(ObjY,VobjY,[VeWHAT]),
prop(of,_,[VexistWHAT,VeX])

And the pattern of *Y has a X of ANSWER* is:

Answer Pattern:
dep(ANSWER,ANSW,[VeANSW]),
prop(of,_,[VeY,VeANSW]),
object(ObjX,VobjX,[VeX]),
evt(have,_,[VeX,VeWHAT]),
object(ObjY,VobjY,[VeY])

Borrowing Prolog notation, the above patterns use uppercase forms or '_' to express the arguments that can unify with logical form arguments. As the logical form of *What is the population of Iceland?* matches the above question pattern, then its logical form is transformed into:

Q: *What is the population of Iceland?*
dep(ANSWER,ANSW,[VeANSW]),
prop(of,_,[VeY,VeANSW]),
object(iceland,o6,[x6]),
evt(have,_,[x6,x1]),
object(population,o4,[VeY])

Now the transformed logical form shares all five terms with the logical form of *Iceland has a population of 270000*, hence the score of this answer sentence is 5. In addition, AnswerFinder knows that the answer is *270000*, since this is the value that fills the slot `ANSWER`.

The use of flat logical forms makes it possible to handle certain types of paraphrases (Rinaldi et al., 2003) and therefore we believe that patterns based on flat logical forms for the task of identifying answer sentences such as the ones described above are more appropriate than patterns based on syntactic information or surface strings. However, the resulting patterns are difficult for humans to read and the process of developing the patterns becomes very time-consuming. Due to time constraints we developed only 10 generic templates. Each template consists of a pattern to match the question, and one or more replacement patterns. The above example is one of the question/replacement patterns.

3.5 Actual Combination of Scores

After experimenting with various ways to combine the lexical, syntactic, and semantic information, the system developed for TREC 2004 used the following strategy:

1. Select the top 100 sentences according to lexical information only (as described in Section 3.3).

2. Re-rank the sentences using the following combinations (several runs were produced for TREC 2004):

   - $3gro + flfo$, that is, three times the overlap of grammatical relations plus once the overlap of flat logical forms. In other words, grammatical relations are given three times more importance than flat logical forms.
   - *flfo*, that is, ignore the grammatical relations overlap and use the logical form information only.

Runs with the formula *flfo* had better results in recent experiments with data from TREC 2005 so we decided to use this formula in the submission to DUC 2006.

4 Sentence Combination

In this work, we aimed to produce a summary composed of extracted sentences. Given a set of questions and a ranked list of returned sentences per question (referred to as an *Answer Set*), the final stage of our summariser determined which sentences were to be selected for the summary. In general, our sentence selection mechanism would return more sentences than could be used in the final summary given the length limit of 250 words per summary.

Our summary content structure was quite simple. For each summarisation test case, we constructed a summary skeleton that allocated an equal portion of the summary for each atomic question derived from the statements in the original DUC topic text. These portions were ordered according to the order of the atomic questions in the original DUC topic (i.e.. question order). Our task then was to populate each of these portions with sentences.

Our overall summary population strategy consisted of the following steps:

1. Perform any necessary re-ranking of sentence lists on the basis of their contribution to the final answer.

2. Pop off the best sentence from each answer set and insert it into the summary portion reserved for the question associated with that list.

3. Repeat step 2 until the summary length limit is reached.

However, in order to populate a summary, we found that we had to handle the following two issues:

Issue 1 *How do we select the best sentences given multiple questions?*

Issue 2 *How do we allocate space to each question?*

In response to Issue 1, our re-ranking of an answer set was designed to handle duplicate extracted sentences across all answer sets. Using the well-established principle in multi-document summarisation (Barzilay and McKeown, 2005), repetition of sentence content was again taken to indicate that the sentence was important. In our case, a sentence that was automatically deemed to answer multiple questions, and which was hence returned several times, was given an elevated score. This was achieved by keeping the first instance of the duplicated sentence (in some answer set) but updating its extraction score (as computed by AnswerFinder) by adding the scores of subsequent repetitions found in later answer sets. These duplicates were then removed. Once the re-scoring based on duplicate detection was completed, sentences in each answer set were resorted in descending order by extraction scores, which are currently integers.[2]

[2]For sentences with the same score, we would have preferred to sort these by conserving the sentence order in the original document. However, our system currently does not return this information to the summary generation module so we cannot ensure this.

To flesh out the summary skeleton, we iterated across answer sets in 'question order'. The top sentence was removed from the answer set and then inserted into the appropriate portion reserved for that answer set in the summary skeleton. Note that as we kept the first instance of a repeated sentence, this sentence tended to appear towards the beginning of the summary. Sentences underwent simple preprocessing before insertion to remove news source attribution information (like 'Reuters __'), or location and time information of the news story prepended to the sentence string.

Our space allocation approach gave space to each question portion opportunistically (see Issue 2) by looking at the quality of the answer sentences extracted as indicated by the extraction score. After one pass across the answer sets, our summary would at least have one sentence answer for each question. However, the 250 word limit may not yet have been reached. Instead of simply filling out the sentences by giving each question equal status (and hence equal portions of the summary), we gave more room in the summary to questions for which our sentence extraction module found good answers. The intuition behind this is that if AnswerFinder has done a better job on some questions and not others, we should not waste summary space by using sentences with low extraction scores, which represent a low match to the question.

To do this, we kept track of the best extraction score seen so far, regardless of which question is being answered. We call this the *Recent Best* score. We then iterated across answer sets and if we found an answer that was as good as the Recent Best score (i.e. equal to it), we appended it to the end of the relevant portion in the summary. If, after examining all answer sets, no sentences were found to be as good as the Recent Best score, we reduced the Recent Best score by one and reiterated again across answer sets. This process of filling in the summary ends when our given word limit is exceeded.

5 Results and Discussion

The results of the evaluation conducted by NIST are shown in Table 6. The table includes the scores for our system, the mean of all the participating systems, the best scores, the worst scores, and the scores of the NIST baseline system. The baseline system returns all the leading sentences (up to 250 words) in the <TEXT> field of the most recent document.

On average, our results are just below the average and above the baseline. The only result below the baseline was the quality score, though it should be mentioned that the baseline score was higher than the best overall score across all systems. Considering the little amount of work spent in the adaptation of the question answering system and the final combination of the answers (under 35 person-hours), these results are encouraging.

We noted that there was not much difference between the scores produced by the automatic evaluations of all the systems. Therefore we did not consider it advisable to evaluate the impact of specific components of our system in the overall results. Instead, we will focus on perfecting the adaptation of the QA system and a more sophisticated way of combining the answers found.

6 Conclusions and Further Research

Our contribution was the result of initial experimentation on the integration of a question answering system for the task of question-driven document summarisation. The results obtained are promising and suggest that there is good potential of such an integration.

The technology used in AnswerFinderis designed to find answers to simple fact-based questions (of the type used in the QA track of TREC). This needs to be adapted to the needs of question-based summarisation. We have identified the following areas of further work:

Processing of the DUC topic description. We plan to attempt a more detailed analysis of the DUC topic descriptions (the <narr> field) so that they are converted into simpler sentences. In some cases this will involve decomposing a sentence with conjunctive elements into several simpler sentences. We currently integrate a dependency-based parser and therefore it would be possible to analyse the sentence dependency structures to perform transformations that would split a dependency structure into several simpler structures.

		Responsiveness		*Automatic Eval.*		
Run	*Quality*	*Content*	*Overall*	*R2*	*SU4*	*BE*
AnswerFinder	3.20	2.40	2.10	0.08	0.13	0.04
Rank	*21-27*	*24-28*	*20-28*	*9-19*	*17-22*	*9-21*
Mean	3.35	2.56	2.19	0.07	0.13	0.04
Median	3.40	2.60	2.20	0.08	0.13	0.04
Best	4.10	3.10	2.40	0.10	0.16	0.05
Worst	2.30	1.70	1.30	0.03	0.06	0.00
Baseline	4.40	2.00	2.00	0.05	0.10	0.02

Table 6: Results of the NIST evaluation; 34 systems participated; the rank range indicates the existence of other systems with same scores

Question analysis. The sentences derived from the DUC topic descriptions system are very different to typical TREC QA questions. Consequently, AnswerFinder's question analyser needs to be modified. Rather than converting the DUC simplified sentences into TREC-like questions, it is probably more useful to develop a taxonomy of sentence types specific to the DUC task, and then to develop a question classifier tuned to these sentence types and patterns. Provided that one can build a training corpus of reasonable size, a good solution would be to use a statistical classifier trained on such a corpus.

We also want to study methods to identify the question focus. Given that the questions are typically in the form of declarative sentences (such as *discuss* . . .), typical methods used to find the focus in questions may not apply here.

Sentence combination. The sentences extracted in the QA component are independent of each other. These sentences need to be re-ranked in order to reduce redundancies and ensure a complete answer.

Summary generation. Our text quality could also be improved further. To make better use of the limited summary length, we will examine methods for compressing sentences. To enhance coherence, we intend to examine schematic orderings of extracted information.

References

Regina Barzilay and Kathleen R. McKeown. 2005. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328, September.

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 2001*, number 500-250 in NIST Special Publication. NIST.

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

Diego Mollá and Mary Gardiner. 2005. Answerfinder at TREC 2004. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proc. TREC 2004*, number 500-261 in NIST Special Publication. NIST.

Diego Mollá and Ben Hutchinson. 2002. Dependency-based semantic interpretation for answer extraction. In *Proc. 2002 Australasian NLP Workshop*.

Diego Mollá. 2001. Ontologically promiscuous flat logical forms for NLP. In Harry Bunt, Ielka van der Sluis, and Elias Thijsse, editors, *Proceedings of IWCS-4*, pages 249–265. Tilburg University.

Fabio Rinaldi, James Dowdall, Kaarel Kaljurand, Michael Hess, and Diego Mollá. 2003. Exploiting paraphrases in a question answering system. In *Proc. Workshop in Paraphrasing at ACL2003*, Sapporo, Japan.

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