Bangor at TREC 2004: Question Answering Track

Terence Clifton, William Teahan
{terence, wjt}@informatics.bangor.ac.uk

ABSTRACT

This paper describes the participation of the School of Informatics, University of Wales, Bangor in the 2004 Text Retrieval Conference. We present additions and modifications to the QITEKAT system, initially developed as an entry for the 2003 QA evaluation, including automated regular expression induction, improved question matching, and application of our knowledge framework to the modified question types presented in the 2004 track. Results are presented which show improvements on last year's performance, and we discuss future directions for the system.

Keywords: Question answering, knowledgeable agents, knowledge grid, regular expressions, regular expression induction.

1. INTRODUCTION

The 2004 evolution of the TREC QA track moved more towards a context based approach to providing the question set. A specified target subject was given, which was then used as the basis for a series of corresponding questions. These questions were made up of three types:

- Factoid
- List
- Other

The TREC QA guidelines [7] describe the 'other' questions as follows:

> The final question in each series is an explicit 'other' question that should be interpreted as "tell me other interesting things about this target I didn't know enough to ask directly". This final question is roughly equivalent to the TREC 2003 QA track's definition question.

Our 2004 entry was a minor evolution of our 2003 system, and focused mainly on automating the creation of further regular expression patterns, which are the key to the generation of our ***Knows*** and ***KnowsAbout*** relations. In addition to the automated regular expression production, we have implemented an improved question matching architecture, and improved the application of our logic framework to the 'list' and 'other' question types.

We describe these modifications, and their effect on system performance in the 2004 QA task, in this paper, which is organised as follows. Firstly we give a general recap of the system architecture which was developed for the TREC 2003 QA task, and provide an overview of the performance in that task in relation to other systems entered (Section 2). We then present the additions and modifications made to the QITEKAT system for the 2004 QA evaluation. In Section 4 we present initial results received from NIST, and a brief analysis of how the system performed. Sections 5 describes our plans for the future development of the QITEKAT system.

2. BACKGROUND

The QITEKAT system is a practical implementation of our logic-based framework for implementing knowledgeable agents that will become the core component for our multi-agent information retrieval systems. A brief overview of the framework and the system architecture is included below, and we refer the reader to [1] for a more detailed explanation.

In [5], we describe a framework for designing and implementing knowledgeable agents and Knowledge Grids. The framework is based on three types of knowledge relations: ***Knows***, ***KnowsAbout***, and **KnowledgeableAbout**. These are used to define what an agent knows, what it knows about, and whether an agent has been judged to be knowledgeable by other agents. Essentially, the architecture is based on using knowledgeable agents as a middle layer between the user and the information resources. A key aspect of the design is the use of information extraction coupled with compression-based language modelling technology [6] and the use of a conversational agent that the user asks questions of and receives answers from the system. In this architecture, there are three types of objects: users, knowledgeable agents and information resources. The users do not interface directly with the information resources. Instead, they must go through a knowledgeable agent who effectively acts as a knowledge broker in determining which of the information resources are likely to contain an answer to the user's questions. Notice that knowledgeable agents may need to go though

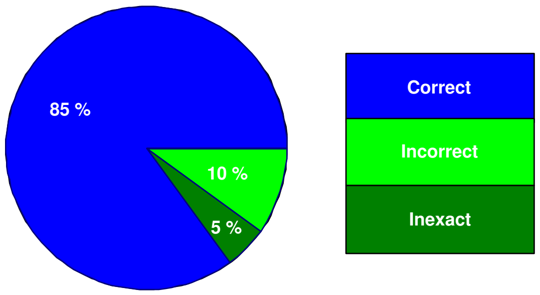

Figure 1. Results on TREC 2003 (*Who* and *When* questions)

other knowledgeable agents in the hunt to find the most relevant answer to the user's questions.

The knowledge framework proposed by Teahan [5], which is used as the basis for the extraction of knowledge relations from suitable source documents essentially relies on a reverse approach to standard Q&A techniques. Rather than using the question text to retrieve a subset of documents from the test collection, which are then analysed to find an answer, the QITEKAT system was designed to parse the entire collection, forming a number of question/answer relations before any actual questions are posed. The TREC Q&A Track uses the AQUAINT document collection as its source corpus, which consists of over 1 million documents, totalling 375 million words. The system was developed based around three main stages:

- Documents are normalised;
- Knowledgeable agents tag and extract Question/Answer pairs;
- Input questions are analysed, and answers are retrieved and ranked.

Figure 2 shows the component make up, and how each of the individual modules interacts with the rest of the system.

Overall performance of the QITEKAT system on the TREC2003 evaluation was satisfactory - table 1 shows the final results for the factoid component evaluation [8]. The initial development timescale meant that the system was designed to only account for two distinct question types (Who and When). Manual examination of the 2003 test questions showed that the system was equipped to deal with a possible 124 questions, of which it correctly answered 107, giving an accuracy score of 86.2% (Figure 1) [1].

Submitter	**Accuracy**
Language Computer Corp.	0.700
LexiClone	0.622
National University of Singapore (Yang)	0.562
University of Southern California, ISI	0.337
IBM Research (Prager)	0.298
Massachusetts Institute of Technology	0.295
University of Wales, Bangor	0.259
University of Albany	0.240
ITC-irst	0.235
BBN	0.208
Fudan University	0.194
NTT Communication Science Labs	0.150
MITRE Corp.	0.148
Chinese Academy of Sciences (CAS-ICT)	0.145
University of Amsterdam	0.145

Table 1. TREC 2003 evaluation scores for the runs with the best factoid component.

3. MODIFICATIONS AND ADDITIONS

In this section we describe the additions and modifications we made to the QITEKAT system for the 2004 TREC evaluation. Minor bug-fixing and optimisation was carried out to improve performance, but the majority of development effort went into the changes detailed below.

3.1. Regular Expressions

Regular expressions were developed to pattern match sentence construction for common question types. This approach is similar to that used by Ravichandran and Hovy in [3]. It was important to make the best use of the previously tagged documents, and to ensure that regular expressions used by the system were not too specific as to require multiple expressions for a single question construct. This led us to develop a dynamic substitution system, whereby a generic regular expression was populated at runtime using the tagged contents of the sentence it was being applied to. We maintained a data store of basic regular expression formats, suitable substitution types, an allowable answer type, and a generic question format for the particular relation.

By using the named entities already tagged in the document, the system can create a number of actual regular expressions, substituting suitable types into the *ANSWER* and *OBJECT* locations. For example, given the sentence:

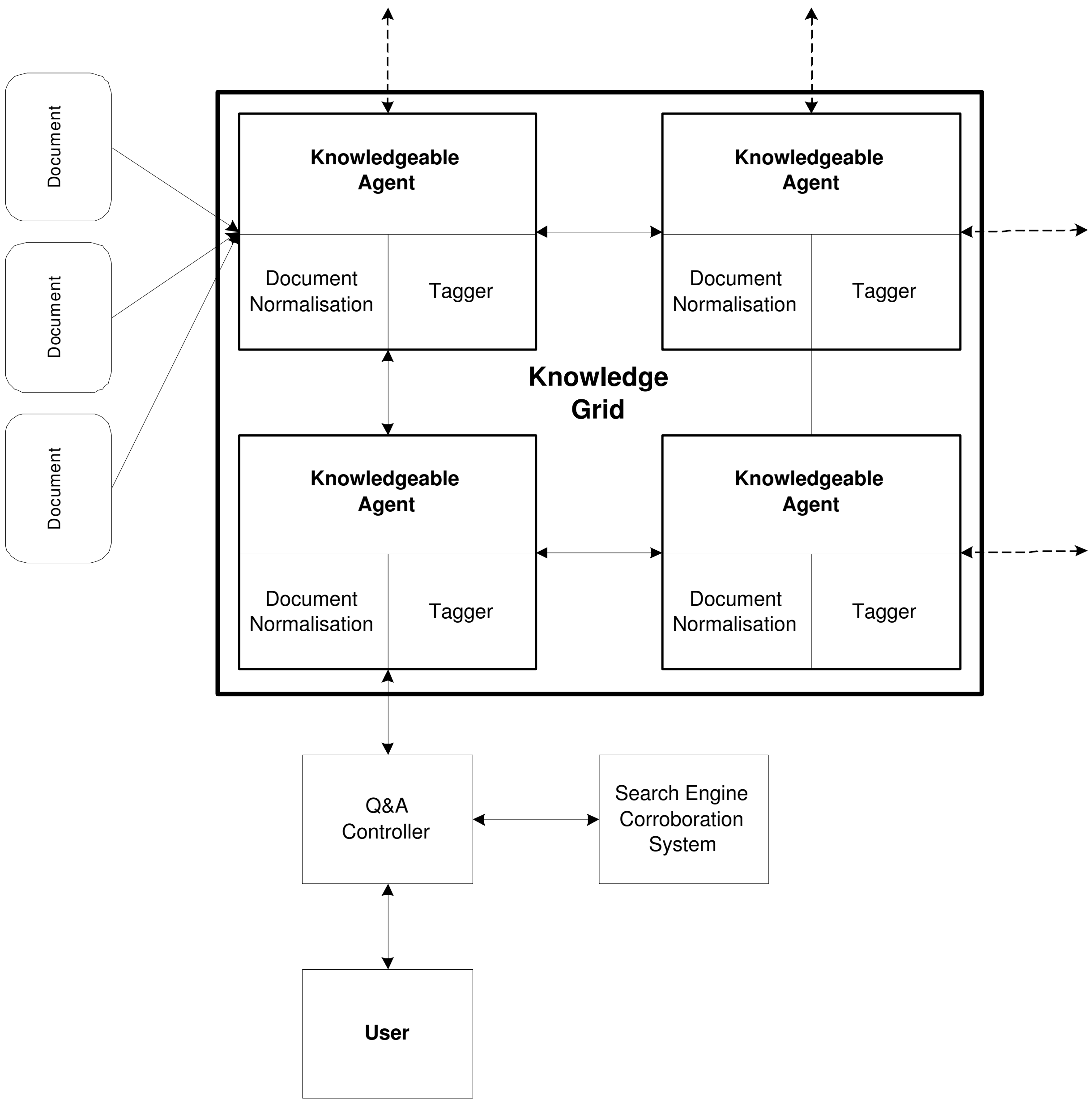

Figure 2. System architecture (simplified)

```
<domain>PEOPLE</domain>
<answer>DATE</answer>
<object1>PERSON</object1>
<object2>NONE</object2>
<object3>NONE</object3>
<regexp>
(OBJECT1)\sdied\s(on|in|around)\s(ANSWER)
</regexp>
<format>When did OBJECT1 die?</format>
```

Figure 3. Example of hand-crafted RE

> “John Lennon died on December 8th, 1980 during a public dramatic interpretation of J.D. Salinger’s Catcher in the Rye"

the system would tag one *DATE* entity (December 8th, 1980) and two *PERSON* entities (John Lennon and J.D. Salinger). The system would then dynamically produce two regular expressions:

1. (John Lennon)\sdied\s((on|in|around)\s(December 8th, 1980)
2. (J.D.Salinger)\sdied\s((on|in|around)\s(December 8th, 1980)

These would then be applied to the document to extract any matches which would be transformed into ***Knows*** relations. In this case, option 1 would match, resulting in the following relation (given that the knowledgeable agent who produced the document text referred to as Doc-XXX-Agent).

```
Knows(Doc-001-Agent, "Domain: PEOPLE",
"When did John Lennon die?",
"December 8th, 1980",
1.0).
```

Further examples of extracted ***Knows*** relations:

```
K1 = Knows(Doc-004-Agent, "Domain: PEOPLE",
"Who is George W. Bush?",
"United States President",
1.0).

K2 = Knows(Doc-004-Agent, "Domain: PEOPLE",
"When was George W. Bush born?",
"July 6th 1946",
1.0).
```

```
<domain>PEOPLE</domain>
<answer>PERSON</answer>
<object1>PERSON</object1>
<object2>NONE</object2>
<object3>NONE</object3>
<regexp>
(OBJECT1)\swife\sof\s(ANSWER)
</regexp>
<format>Who is OBJECT1 married to?</format>
```

Figure 4. Example of auto-generated RE

These Knows relations are then used to populate suitable ***KnowsAbout*** relations such as the following:

```
KnowsAbout(Doc-004-Agent, "Domain: PEOPLE",
"George W. Bush",
{K1,K2},
1.0).

KnowsAbout(Doc-001-Agent, "Domain: PEOPLE",
"John Lennon",
Ka,
1.0).
```

A small number of broad domain types were used (*PEOPLE*, *GEOGRAPHY*, *BUSINESS*, *MISC*).

Our initial regular expressions were hand-crafted, but it became quickly evident that this would not be efficient, either in terms of the time taken, or the required generality of the expressions. Using previous Question-Answering data as a source, we were able to implement an automated system to generate regular expressions, based on a combination of entity type tagging and proximity matching. Given a source document, a question and an answer (that exists in the document), the following procedure is followed:

1. Extract the ‘subject’ of the question using traditional speech tagging techniques, and PPM compression based language modeling (see [6] for further details).
2. Analyse the source document for the proximity of the known answer to the subject.
   (a) Partial matching is applied here to ensure that subjects are recognised in the answer document.

Proximity	No of Regexps	No of Questions
10	152	1.4 million
20	263	2.8 million
50	393	3.2 million
100	469	3.6 million
150	532	3.7 million
200	566	3.8 million
500	579	3.9 million

Table 2. Effect of proximity level on regular expression generation

3. If the answer falls within a given proximity threshold to the subject (i.e. is within a certain number of characters either side of the subject), we retrieve the surrounding subtext.
4. This subtext is then parsed and a regular expression generated.
   (a) Stopwords are ignored.
   (b) Named Entity tags are inserted where possible to generalise the regular expression.

Our experiments with different proximity limits (number of characters) on the AQUAINT corpus led us to adopt a proximity level of 150 characters, which offered the best compromise between performance and the quality of expressions. (Larger proximity expressions lose generality, and thus effectiveness).

3.1.1. Example

Given the following information:

Source Document	NYT19980601.0001
Known Question	When did Kenneth Lenihan die?
Known Answer	May 25
Source Excerpt	Kenneth Joseph Lenihan, a New York research sociologist who helped refine the scientific methods used in criminology, died May 25 at his home in Manhattan. He was 69.

Table 3. Example source information for regular expression induction

We would extract a question subject of 'Kenneth Lenihan', and parse the source text to find a suitable match. Through the use of our partial matching algorithm [1] we are able to recognise a match between 'Kenneth Joseph Lenihan' in the source text and 'Kenneth Lenihan' as our subject, resulting in a matching string:

> "Kenneth Joseph Lenihan, a New York research sociologist who helped refine the scientific methods used in criminology, died May 25"

From this string we apply our proximity check to determine if the match is within a suitable distance. In this example, the subject is 101 characters from the answer, and thus the match is accepted. We then generalise the string to a suitable regular expression, by removing stopwords and inserting named entity classes where appropriate. Part-of-speech groups in close proximity to the answer, which correlate to the question text are kept to ensure the meaning is retained:

```
"PERSON\s.*\s(LOCATION\s)?
(PROFESSION\s)?.*\sdied\sDATE"
```

This would provide us with the regular expression construct shown in figure 5:

```
<domain>PEOPLE</domain>
<answer>DATE</answer>
<object1>PERSON</object1>
<object2>LOCATION</object2>
<object3>PROFESSION</object3>
<regexp>
(OBJECT1)\s.*\s((OBJECT2)\s)?▷
((OBJECT3)\s)?.*\sdied\s(ANSWER)
</regexp>
<format>Who is OBJECT1 married to?</format>
```

Figure 5. Generated regular expression

3.2. Improved Question Matching

Upon completion of the 2003 evaluation, it was clear that in a number of cases where incorrect, or no answer was returned by the system, this was not because the answer was not found, but due to an inability to cross-match the entered question with the question in the knowledge base. The original system used a very limited string matching system, coupled with identification of question types (*What*, *When*, etc). Unfortunately, even small discrepancies in entered and known questions would result in a NIL match. The 2003 evaluation proved that this was insufficient, and that a better system was needed.

We implemented a vector matching system, whereby entered and known questions were compared based on their non-stopword content. In addition to simple positional matching, we added two extra factors that influenced match-weight:

- Subject - Matching the subject explicitly (i.e. not a partial match, see below) boosted a question's match-weight.
- Word frequency - Using the same dictionary we adopted for tagging, we tested matched words for frequency, boosting weights for less frequent words, as a match was likely to have more significance.

To handle the changes brought in to the question format for the 2004 task regarding subject specification, we implemented a simple system to substitute the subject for occurrences of personal pronouns in the question text, and applied our partial matching algorithm [1] to provide greater generality to the produced questions.

3.3. Handling 'Other' and 'List' Questions

The knowledge framework which forms the basis of the QITEKAT system was designed to incorporate the notion of context, and this was practically implemented through the use of domain specification both at the ***Knows*** and ***KnowsAbout*** level of our architecture. This gave us an excellent starting point when addressing the 'list' and 'other' question types which formed part of the 2004 TREC QA evaluation. In its original incarnation, the QITEKAT system specified very broad domain constructs for each question it extracted from the source corpus (*PEOPLE*, *GEOGRAPHY*, etc), which were then grouped within each agent to form our knowledge base. These were generated using a two-step approach:

1. Determine the most likely question/answer subject. This was done using traditional speech tagging techniques, and PPM compression based language modeling (see [6] for further details) to extract named entities from the question. When required, selection was made based on frequency of occurrence in the parent document, with the assumption that more frequent occurrences were likely to be the focus of information.
2. Our named entity tagger was then applied to this 'focus' object to determine a broad domain classification (PERSON entity type yields PEOPLE domain, etc.)

The new format, with explicit specification of the subject of the question, correlated well with our method, as we were simply able to remove our second-step, and store the named entity as our domain classification. These specific domains were then grouped into KnowsAbout relations stored at each agent and generalised where possible using our partial matching algorithm. For example:

- *Bush*;
- *George Bush*; and
- *George W. Bush*

occurring in three different ***Knows*** relations in the same document would be grouped into a single ***KnowsAbout*** relation. In this case, the more explicit domain - George W. Bush - would be used, to allow for improved matching at later stages in the process.

Once we had our ***KnowsAbout*** information, answering 'list' queries was a matter of retrieving all corresponding relations for a particular question subject, applying our improved question-matching algorithm, as detailed previously, to find answers that correspond to those required by the list, and returning the results. In generating answers to the 'other' questions we would have ideally liked to reconstruct useful 'nuggets' from the question/answer pairs our system extracted, but time didn't permit this element of the system to be completed, and it has been scheduled for a future revision. As a result we were only able to return our known answers, and provide no context from the question, which made the information of little use (in fact, technical issues meant no answers were returned at all - see results section).

3.3.1. Example

Given the following question/answer pairs extracted from a document:

- When was John Lennon born? *October 9, 1940*
- When did Lennon die? *December 8, 1980*
- How did John Lennon die? *Assassinated*

The system would first extract the named entities from each of the questions to determine their subjects

- John Lennon
- Lennon

- John Lennon

And produce the ***Knows*** relations:

```
K1 = Knows(Doc-001-Agent, "Domain: John Lennon",
"When was John Lennon born?",
"October 9, 1940",
1.0).

K2 = Knows(Doc-001-Agent, "Domain: Lennon",
"When did Lennon die?",
"December 8, 1980",
1.0).

K3 = Knows(Doc-001-Agent, "Domain: John Lennon",
"How did John Lennon die?",
"Assassinated",
1.0).
```

Using partial matching, the system would recognise 'Lennon' and 'John Lennon' as the same subject class, and produce the following ***KnowsAbout*** relation (using the more explicit subject as the topic, and the general entity type as the domain):

```
KnowsAbout(Doc-001-Agent, "Domain: PEOPLE",
"John Lennon",
{K1,K2,K3},
1.0).
```

4. DISCUSSION

Performance of the QITEKAT system in the 2003 evaluation was a useful benchmark on which to build, and initial results made available by NIST for the 2004 task show that gains have been made in both 'factoid' and 'list' categories, as well as in the overall f-score. The 2003 results were hindered by the limited development time, which meant regular expressions were only created for a small subset of question types. This years' performance reflects the addition of the automated expression system, and the corresponding increase in the number of regular expressions generated, and thus the question types covered. Factoid accuracy moved from 26% in 2003 to 64% in 2004. 'list' and 'other' (definition) questions were adversely affect by technical issues in 2003, meaning a zero score was recorded for both types. These problems were addressed for the 2004 evaluation, producing a list score of 0.258, but unfortunately, further problems manifested themselves, producing NIL answers for all 'other' questions. Overall, the system performance has progressed from a combined F-score of 0.130 in 2003, to 0.388 this year. Figure 6 shows a graphical representation of performance over the two years that Bangor has participated in the evaluation.

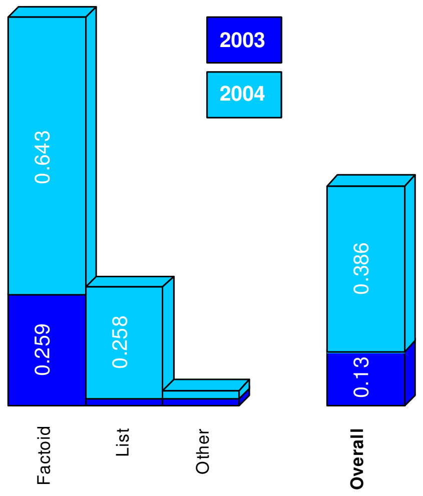

Figure 6. Performance comparison 2003-2004

These preliminary results show a good performance gain for the QITEKAT system, with a 38% improvement in factoid scores, and an almost three-fold increase in overall performance, which is very promising. Elimination of some technical issues which meant the 2003 evaluation was not a complete run of the system, can account for some of the gains (although the 2004 run suffered its own problems which limited the analysis time also), while auto-generating the regular expressions rather than hand-crafting them was far more efficient in terms of time, and meant that a greater proportion of question types could be targeted and answered by the system.

Although results for other groups entered in the current evaluation will not be made available until after the TREC conference, based on the known min/max/average scores for each of the question types (Factoid, List, Other) we can make a preliminary assessment as to the performance of the QITEKAT system in relation to the other entries. Table 4 shows the relative comparison of the QITEKAT performance with the know minimum, maximum and average scores for the 2004 evaluation*. These figures imply that factoid

*Overall minimum, maximum and average scores are not provided by NIST and are estimated by assuming that the MIN/MAX/AVERAGE in each category are from the same system run.

performance on the 2004 evaluation was very good - well above average, and the 'list' performance was also above the recorded. The technical issues with 'other' questions resulted in a zero result, which pulled down the overall score. This was still above average, however, and a good improvement on last year.

	QITEKAT	MIN	MAX	AVE
Factoid	0.643	0.009	0.770	0.170
List	0.258	0.000	0.622	0.094
Other	0.000	0.000	0.460	0.184
Overall	0.386	0.005	0.656	0.155

Table 4. Relative comparison of 2004 evaluation results.

5. FUTURE DIRECTIONS

A number of additions and modifications have been recognised which could provide improved performance, and are outlined below.

5.1. Improved Named Entity Classification

Although the Named Entity classifier developed as part of the system performs well, for the purposes of Question Answering it is important to broaden the scope of the system, and introduce further NE types in order to allow for more accurate answer matching. Sekine *et al.* present a system offering a far greater number of Named Entity classifications [4], which we feel would be a beneficial addition to the overall system architecture.

5.2. Synonym substitution

The present system architecture offers no methods for word substitution, which is a limiting factor, both in terms of matching questions with appropriate knowledge relations, and also extracting relations from document texts. The addition of a synonym system, such as WordNet [2] would enable a greater number of sentence constructs to be identified and extrapolation of questions to form multiple queries, offering a far greater chance of successful responses. As an example, take the question text:

"When did Charles Bronson die?"

In the present system, this will match only those relations with an equivalent question construct, which may result in no answer being found. With synonym substitution, however, the query would be reformulated as:

"When did Charles Bronson pass away?"

which may provide a positive match.

5.3. Past Participle Determination

In a similar vein to synonym substitution, it would be useful to develop a feature within the system to automatically generate past participles of verbs, particularly for confidence-ranking through search engine corroboration. When querying a search engine, the system passes the main subjects of a question, so for example, given the question:

"When did Charles Bronson die?"

The system forms a query using ***Charles Bronson*** and ***Die***. It is likely however that in any documents retrieved by a search engine the information that we are interested in would be described using the past participle (**died**), i.e.

"Charles Bronson died on .."

Substituting the past participle may result in a more useful query string, and ultimately a greater number (or more accurate) results.

5.4. Automated Regular Expression Production

Further development of our automated regular expression system is required to handle multiple subject questions, such as:

"How far is it from New York to Los Angeles?"

"Is New Zealand larger in size than Japan or Great Britain?"

We are investigating the feasibility of parsing input into two or more atomic questions, which can then be addressed and the answers combined to formulate a response to the original query. This would allow us to extract further information from the source corpus, and extend the number of question types the system is capable of answering.

5.5. 'Other' Questions

We are currently investigating the problems encountered and the conversion our Question/Answer pairs into useful 'nuggets' of information about specified question subjects. We hope to be able to leverage the fact that we explicitly store domain information for all question/answer pairs extracted from the corpus in order to target this particular problem successfully.

REFERENCES

[1] Clifton T., Colquhoun A., Teahan, W. "Bangor at TREC 2003: Q&A and Genomics Tracks". In *Proceedings of the Twelfth Text Retrieval Conference*, 2003, NIST. pp 600-611

[2] Miller G. "WordNet: An On-Line Lexical Database". In *International Journal of Lexicography.* 2002

[3] Ravichandran D., Hovy E. "Learning Surface Text Patterns for a Question Answering System". In *Proceedings of ACL* 2002, pp 41-47.

[4] Sekine S., Sudo K., and Nobata C. "Extended Named Entity Hierarchy". In *Proceedings of the LREC-2002 Conference*, 2002. pp 1818-1824.

[5] Teahan W. 2003. "Knowing About Knowledge: Towards a Framework for Knowledgeable Agents and Knowledge Grids". *Artificial Intelligence and Intelligent Agents Tech Report AIIA03.2*, School of Informatics, University of Wales, Bangor.

[6] Teahan W., Harper D. "Using Compression-Based Language Models for Text Categorization". In *Language Modelling for Information Retrieval*, 2003, Kluwer Academic Publishers. pp 141-165.

[7] Vorhees E. Q&A Track Guidelines. *Thirteenth Text Retrieval Conference*, 2004, NIST. `http://trec.nist.gov/act_part/tracks/qa/qa.04.guidelines.html` (restricted)

[8] Vorhees E. "Overview of the TREC 2003 Question Answering Track". In *Proceedings of the Twelfth Text Retrieval Conference*, 2003, NIST. pp 54-68.