Bangor at TREC 2003: Q&A and Genomics Tracks

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

This paper describes the participation of the School of Informatics, University of Wales, Bangor at TREC'2003 in the Q&A and Genomics Tracks. The paper is organized into three parts as follows. The first part provides a brief overview of the logic-based framework for Knowledgeable Agents that is currently being developed at Bangor. This was adopted as the basis for implementations used for both Tracks. The second part describes the Q&A system that was developed based on the framework, and the final part describes some experiments that were conducted within the Genomics Track at specifying context using GeneRIFs (for a Q&A system being developed for the BioMedical domain).

"Knowing About" Knowledge: A Framework for Knowledgeable Agents

A.1 Introduction

We are in the process of designing and developing a novel logic-based framework for implementing knowledgeable agents that will become the core component for our multi-agent information retrieval systems.

In Teahan (2003), 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. In Teahan (2003) and in the second part of this paper, we describe how a Knowledge Grid could be implemented (based on the framework) which has "knowledge" based on the three defined knowledge relations. 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 (Teahan & Harper, 2003) 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 other knowledgeable agents in the hunt to find the most relevant answer to the user's questions.

A.2 A framework for knowledgeable agents

This section outlines the logic-based framework that we wish to use as the basis of knowledge within the Knowledge Grid architecture. We wish to stress that the framework as described below is still in its developmental stage, and its final form, we envisage, will be somewhat different based on the experiences we garner from future research.

We feel that the traditional propositional truth-based approach that epistemic logic-based multi-agent systems take, which are usually formulated as normal modal logics using the semantics of Kripke (Wooldridge, 2002), is not sufficiently expressive enough for our purposes. Instead, we would like to adopt some of the capabilities of Question/Answering systems within our inference capabilities. A problem with the propositional truth-based approach is that although we can state what an agent may know *per se*, it does not help us find out whether an agent *knows* an *answer* to a *question*, and just as importantly, *what answers* an agent *knows* to a *question*. Neither does it help us find out what an agent knows *about* (where knowing *about* a topic implies that you know *something* about the topic, but it does not imply that you know *everything* about the topic).

We feel that there are three necessary conditions for an agent to be "knowledgeable". The key condition, which we refer to as the *Knowledge Test*, is the following: "*An agent is judged to be knowledgeable by other (external) knowledgeable agents*". This states that judges are used to adjudicate on whether an agent is knowledgeable or not (analogously to the Turing Test in Artificial Intelligence). The judges are agents – either human or computer-based – that must also be "knowledgeable". Like the Turing Test, it is assumed that a question and answering testing process is used before making the judgment. The second condition is a logical consequence of the first condition: "*Other agents must have the ability to learn about and/or be informed of what the agent knows about.*" Simply stated, if other agents don't know about what the agent knows about, then they can't make a judgment in the first place. The third condition states: "*The agent must know: what it knows about, and what it doesn't know about.*" This again relies on the judging process used for the Knowledge Test: it would seem a natural response for a knowledgeable agent to answer "*Sorry, I only know about X and not Y*" to something it doesn't know about.

We have devised the following logic-based framework based on these conditions. We define three logical relationships – *Knows*, *KnowsAbout* and *KnowledgeableAbout*.

A.2.1 The *Knows* relation

We define *Knows*, a 5-tuple relation, as follows:
Knows (*agent, context, question, answer, relevance*).

This is explained as follows: The specified *agent* believes that an *answer* to a *question* for a specified *context* has the specified *relevance* (this is a real number in the range 0 to 1.0 with 1.0 indicating absolute belief that the answer is relevant to the question). A *representation* of the context, question and answer is provided by the specified *context, question* and *answer* which can be arbitrary text passages or strings or some other representation (depending on the implementation). Note that it is possible to ask the same question but in different contexts. The following example is provided as further explanation.

Example 1: *Knows* (*A*, *"Domain: Geography"*, *"Where is Bangor?"*, *"North Wales"*, 1.0).

In this example, the agent believes she knows that the answer to the question "*Where is Bangor*?" is "*North Wales*". She assigns a relevance ranking of 1.0 (in other words, she believes that the answer is certainly correct). The context in this case is the domain of geography.

An agent may believe many answers are relevant to a particular question and context. As a shorthand notation, we write this in the following manner:
Knows (*agent*, *context, question*): *answer*$_1$, r_1; *answer*$_2$, r_2; …

We can also assign an agent's list of answers to a variable. For example, K_A = *Knows* (*A*, *"Domain: Geography"*, *"Where is Bangor?"*). Similarly, we can assign to a variable all that an agent knows on a particular context: K_B = *Knows* (*B*, *"Domain: Geography"*).

A.2.2 The *KnowsAbout* relation

We define *KnowsAbout*, a 5-tuple relation as follows:
KnowsAbout (*agent*, *topic, context*, *knows, relevance*).

This is explained as follows: The *agent* believes that the list of questions and answers denoted by *knows* are related to the *topic* given the *context* and have the specified *relevance*. Intuitively, the agent believes that she *knows about* the topic given a certain context because she knows the answers to the specified questions. Topics and contexts can be arbitrary text passages or strings as above for the *Knows* relation. Note that the agent may know about the same topic but in different contexts.

Example 2: *KnowsAbout* (*B*, "*Bangor*", "*Domain: General Knowledge*", K_A, 0.9).

In this example, agent *B* has some general knowledge about the topic "*Bangor*". What he knows are the same answers that agent *A* knows to the question "*Where is Bangor*?" in a geographical context. He assigns a weighting of 0.9 to his belief in the relevance of agent *A*'s answers.

It seems reasonable to assume that if an agent knows the answer to something, then it knows *about* that something. This is written as follows:
$\forall$*agent, context, question, relevance*
K_i = *Knows* (*agent*, *context, question, answer, relevance*) $\Rightarrow$ *KnowsAbout* (*agent*, *question*, *context*, K_i, *relevance*).

In this case, the agent is inferred to know *about* each question because she knows the answers to them. So for example, from K_A above, we can infer that agent *A* knows about the following: *KnowsAbout* (*A*, "*Where is Bangor?*", "*Domain: Geography*", K_A, *relevance*).

By default, the same relevance from the *Knows* relation can be adopted for the *KnowsAbout* relation, although this can be overridden at a latter time.

It also seems reasonable to assume that if an agent knows the context of a given question and answer, then it knows *about* that context. This is written as follows:
$\forall$*agent, context, question*
K_i = *Knows* (*agent*, *context*, *question, answer, relevance*) $\Rightarrow$ *KnowsAbout* (*agent*, *context*, *context*, K_i, *relevance*).

In this case, the topic that the agent knows about is the context itself.

A.2.3 The *KnowledgeableAbout* relation

We define *KnowledgeableAbout*, a 6-tuple relation as follows:
KnowledeagbleAbout (*knowledgeable-agent*, *testing-agent*, *agent*, *topic, context*, *relevance*).

This is explained as follows: The *knowledgeable-agent* believes that the *agent* is knowledgeable about the *topic* given the *context* with the specified *relevance* because that agent knows about the same things as the *testing-agent* knows about. Effectively, an external agent, which is designated as being knowledgeable, uses test questions to determine if a person or agent knows about some topic. The knowledgeable agent delegates the testing agent to perform the test – this may be a "virtual" agent that is provided with sufficient knowledge necessary in relation to the test. The testing agent may in fact be provided with a subset of the knowledge known by the knowledgeable agent. Alternatively, other possibilities are having the knowledgeable agent spawn a testing agent to perform the test, or having the knowledgeable agent designate an independent testing agent to perform the test. Notice that the series of test questions have themselves now become a form of knowledge.

A.3 Questions, contexts and topics

Note that in our definitions above of *Knows*, *KnowsAbout* and *KnowledgeableAbout*, we do not explicitly state how the questions are represented or how they are to be matched with each other, and similarly for the topics and contexts. If questions, contexts and topics consist of text strings, an inference system may simply impose the *strictest* requirement that the strings match exactly. Alternatively, a less strict matching system may be employed. For example, if contexts are specified as a *set of labels* that specify the context's relevant domains then the conditions for contexts to match may simply be that there exists at least one label common to both sets of domains. For example, the context for the question "*How do you cook pumpkin pie*?" may be the set of domain labels "*Domain: Cooking, Recreation*". Another agent may know about the answer to the same question, but in the slightly different context, "*Domain: Cooking, Hobbies*", although the inference system may infer the contexts match because of the common label "*Cooking*" present in both.

We have deliberately left open the specific representation of the questions, contexts and topics to the designer of the knowledge system. We feel that different representations are required in different applications depending on the nature of the knowledge that needs to be specified and/or manipulated. For example, in a knowledge-based information retrieval system, the context could be used to specify the *information purpose* of the agent that produced each document, and then this can be matched against the *information need* of the user based on the user's question and previous questions.

A.4 References

W. Teahan and D. Harper. 2003. "Using Compression-Based Language Models for Text Categorization" In *Language Modelling for Information Retrieval,* Chapter 7, pp 141-165. Kluwer Academic Publishers.

W. Teahan. 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.

Wooldridge, M. 2002. *An Introduction to MultiAgent systems*. Wiley, New York.

QITEKAT
Question Inference Tools Employing Knowledgeable Agent Technologies

Abstract

We present the QITEKAT Question-Answering system based on the conceptual theory of *Knowing About Knowledge*, which adopts an agent-based approach to extract information from suitable corpora. The components of the QITEKAT system entered by the School of Informatics, University of Wales, Bangor, in the 2003 Text Retrieval Conference are described in detail. We describe PPM compression techniques for Named Entity classification; distributed agent technologies for developing a Knowledgeable Framework and Knowledge Grid; and a Search Engine corroboration system for generating confidence estimates for Question Answering. We present favourable results for certain question types in the TREC Question Answering Track, and discuss future directions for the QITEKAT architecture.

B.1. Introduction

In developing the QITEKAT system, we were aiming to take a first step on the TREC road, providing a foundation for future development within the School of Informatics, at the University of Wales Bangor, for knowledge representation, extraction and language processing techniques. Agent technologies and techniques are a popular tool in modern computer science, and have been applied to a number of problems, including previous TREC question and answering tracks (Chu-Carroll et al, 2002). As a secondary goal, we were aiming to use the TREC Question Answering track as a benchmark to evaluate a developing framework for Knowledgeable Agents and Knowledge Grids (Cannataro and Talia, 2003), based on the concepts of 'Knowing About Knowledge' (Teahan, 2003).

The short development time period (7 weeks) meant that many of the core components of the QITEKAT system were based on standard information extraction and question/answering techniques, although we were able to incorporate a number of interesting features, particularly in Named Entity tagging and relevance ranking.

This report firstly describes the main components of the UWB QITEKAT Question Answering System (Section 2). Section 3 presents results obtained from various experiments on past and current TREC Q&A data, and is followed by a brief analysis of the performance of the system (Section 4). The report concludes with a discussion of possible future enhancements (Section 5).

B.2. System Description

The QITEKAT system was designed not only to offer a practical implementation for the theoretical concepts of 'Knowing About Knowledge', which are explained in greater detail at the start of this paper, but to offer a foundation for the future development of information extraction and question answering techniques to enhance the system for future use, either through the TREC forum, or for practical applications. This need for extensibility, and to be able to swap out various sections of the system as new techniques were developed, leant itself to the use of an object-oriented development platform. We decided that the Java language would offer the greatest flexibility for future development.

The knowledge framework proposed by Teahan (Teahan, 2003), 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 2003 Q&A Track uses the AQUAINT document collection as its source corpus, which consists of over 1 million documents, totalling 375 million words. Quite obviously, performing any kind of extensive parsing or analysis of this size of document collection would be computationally intensive, and not best suited to the Java language.

With these considerations in mind we adopted a 2-level modular approach to the system development, using the Java language to facilitate extensibility, and C where speed was of the essence, integrated using Java native methods. The system was developed based around three main stages:

- document normalisation and storage;
- knowledgeable agents;
- question analysis and answer ranking.

Figure B.2 shows the component make up, and how each of the individual modules interacts with the rest of the system, and a more detailed explanation of each of the key components follows.

B.2.1 XML Document System

Although the TREC Q&A track was our main target during the development of the QITEKAT system, it was important that we consider its application to other areas and document sources. With this in mind we developed a rudimentary XML notation to normalise any source documents, and store them in a consistent fashion for analysis by the Knowledgeable Agents of the system.

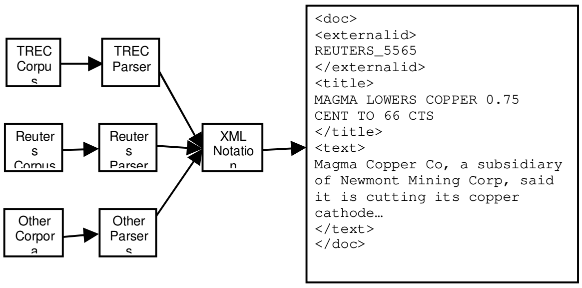


Figure B.1 – Simple XML Notation

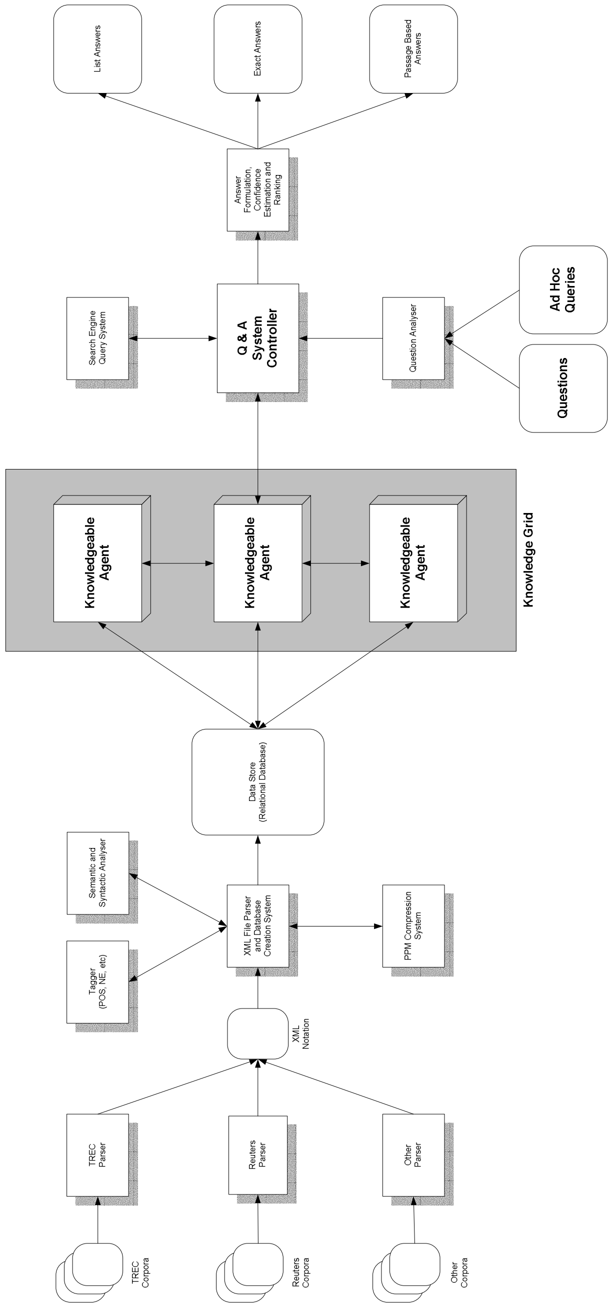


Figure B.2 – System Design

This means that the addition of a new corpora or alternative source of information could be handled using a simple Java based API, and plugged into the system as details of the data source become available.

B.2.2 Speech Tagger

The speech tagger forms a major portion of the QITEKAT development, as the part of speech and named entity tags are used as the basis for extracting knowledge relations from the AQUAINT documents.

The system is loosely based upon the Fastus system (Hobbs et al, 1996), employing the architecture of a cascading finite state automata in order to achieve usable levels of performance. Each stage of the system was developed as a switchable module, so it could be invoked as required, depending on the document structure it is being used to parse.

The system handles each document as a complete entity, separating it into sentences, and then words, before passing it on to first the POS tagger, and subsequently the NE tagger. Again, XML is used extensively to provide run-time modification of the rules and constructs used to tag the portions of a sentence.

POS Tagger and Phrase Chunker

The Part of Speech (POS) tagger is a 2-phase tagger, adopting ideas proposed by Brill (Brill, 1992). It uses a frequency count, extrapolated from a pre-tagged version of the Brown corpus to assign preliminary part of speech tags to each word in a sentence, using the Penn Treebank tagset. These pre-tagged words are then re-examined by a transformation based tagger. The rules for this tagger were developed through automatic examination of the Brown corpus, with some minor manual modification.

Once tagging of individual parts of speech is complete, the sentences are passed on to the phrase chunking module, which adopts a three-stage approach. A POS is tagged with one of three standard types

- **I**nside a chunk;
- **O**utside a chunk;
- **B**oundary of a chunk.

These are based solely on the POS tag assigned to the particular word. This phase is succeeded by a transformation based chunk, again based on rules generated from the Brown Corpus.

Once the final chunking is complete, each phrase chunk is examined to determine its content, and is labelled accordingly (Verb, Proper Noun, Noun, Punctuation, Other).

XML is used to store the transformation and frequency rules for this portion of the speech tagging system, offering a look-ahead/behind matching system on three entity types:

- Words;
- POS Tags;
- Chunk Tags.

Figure B.3 shows an example of an XML rule description for transforming a noun tag (NN) to a verb tag (VB). We can see that the rule specifies that in order for the transformation to take place the POS tag *TO* must be found in the position before the tag being examined. Multiple conditions can be applied for each rule. The validity of the new tag is checked using the original tag frequency information from the Brown corpus, to ensure that the new tag is a suitable option for the current word. Transformations are applied in frequency order and can be cascaded to apply multiple transformations to the same entity.

```
<rule>
<initialtag>NN</initialtag>
<newtag>VB</newtag>
<condition>POS</condition>
<operator1>TO</operator1>
<operator2>-1</operator2>
</rule>
```

Figure B.3 – XML Based Tag Transformation Rule

NE Tagger

Once phrase chunking and identification is complete the system is aware of the phrases in a document that correspond to Proper Noun phrases, and are therefore candidates for Named Entity Tagging. Each of the phrase chunks is passed to the NE tagger, which applies a cascading series of modules to determine the type of Named Entity that the chunk refers to: Currencies; Dates; Times; Locations; Professions; Relations; Measures; Organisations; Names (Pre/Post Honours).

Each of these types is defined by a series of rules, again stored as XML for easy modification, which rely on a combination of direct matching, designator matching and sure-fire context rules

Direct matching

Certain named entity types fall into this category, in particular dates and times, which follow a series of standard word patterns. Regular expression matching is used to identify matches, which are then tagged accordingly. For example the regular expression below can be used to match the initial portion of a date such as 23rd October.

```
((0?[1-9]|[1|2][\\d]|3[0|1])(st|nd|rd|th)?)
```

Designator Matching

This method is adopted to determine such NEs as organisations and persons, and relies on common pre and post entity word matches. For example if we have the NE *British Gas Plc*, we can match the Plc designator, and tag the phrase as an organisation. Other such designators that the QITEKAT system relies upon are:

Mr	Sr	Corp
Dr	Ltd	Jr

Sure-fire context rules

Certain sentence constructs are used to determine the type of Named Entity for a specific phrase, where the surrounding context unambiguously denotes a specific type. As an example, take the partial sentence:

`Shares in XYZ rose 54% on the days trading…`

The context *Shares in ???* implies that ??? is an organisation, and can be used as a suitable sure-fire rule to tag that particular unknown NE.

A small number of these sure-fire rules were manually created (again stored as XML constructs) in order to tag these particular sentence contexts.

PROFESSION of ???	PERSON
RELATION of ???	PERSON
??? province	LOCATION

Partial Matching

Natural language, and particular the construction of news articles, which are the basis for the TREC Question & Answering Track Corpora, often rely heavily human memory and implicit definition. For example, in an article about a particular person, they may be referred to by their full name only once, early in the article, yet will be referred to again on numerous occasions throughout the text. This may be by some abbreviation of their name, say their surname only, or some other means such as anaphora (*He said…*). It is important for a successful Named Entity tagging system to be able to handle this *cross-reference* in a particular document in order to correctly tag the unknown NEs present.

The QITEKAT system employs a simple partial matching algorithm to solve these problems, and cross-tag equivalent entities. This works by extracting all known NEs from a particular document (which have been identified previously, either by designator matching or some other means) and creating partial orders of each. These partial orders are then compared to the remaining unknown NEs in the document, and should a match occur, the new NE is tagged with the equivalent type.

As an example, take a document that discusses the work of `Dr. Bill Teahan`. This phrase would be correctly identified as a PERSON, by matching of the Dr. designator. Partial orderings of this phrase would then be constructed (retaining word order to ensure correct cross-matching):

`Dr. Bill, Dr. Teahan, Bill Teahan`

Should these phrase constructs occur elsewhere in this same document, they would be tagged according to the original phrase (i.e. As a PERSON type).

PPM-Based Language Modelling

The final stage of the QITEKAT speech tagging system focuses on labelling all remaining unknown NE phrases, and adopts a compression-based language modelling system to achieve this goal. Much research has been carried out into the use of PPM compression systems for the text classification (Teahan and Harper, 2003), whether it be to identify languages, determine authorship or otherwise.

We have adopted a PPM based compression system to deal with unknown NE classification, by training PPM models on various known data sets corresponding to the available NE types in the QITEKAT system (PERSONS, ORGANISATIONS, etc). Given a suitably large data set of known phrases of each type, we have been able to train compression models for each. These models are then used in turn to compress unknown phrases from the document set. The model providing the best compression level (i.e. the shortest code length) is thus assumed to be the most appropriate type for the unidentified phrase.

In initial tests on 200 Reuters news articles, this compression system was able to produce very favourable results, when applied as the final stage in the QITEKAT tagging process.

Number of unknown NEs	141
Number of NEs correctly identified	132
Number of NEs incorrectly identified	9

B.2.3 Knowledgeable Agents

The theory of Knowledgeable Agents proposed in Teahan, 2003, and outlined at the start of this paper is used as the basis for the main document processing component of QITEKAT. Each agent is capable of running autonomously and analysing a given series of XML documents to generate *Knows* and *KnowsAbout* relations, which it then stores for the purpose of question-answering.

B.2.3.1 Regular Expressions

In order to extract *Knows* relations from the AQUAINT corpora, regular expressions were developed manually to pattern match sentence construction for common question types. These expressions were developed using the TREC 2001 question text, and focus on the *Who* and *When* question types only, due to time constraints.

It was important to make the best use of the 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 RE was populated at run-time using the tagged contents of the sentence it was being applied to.

Again all rules are stored in an XML file, to enable rapid updating and maintenance of the rule base, and a typical entry looks as follows. The file denotes a basic regular expression format, suitable substitution types, an allowable answer type, and a question format for the particular relation

- When did OBJECT1 die?
- Who was OBJECT1?

```
<questionpack>
<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>
</questionpack>
```

Figure B.4 – XML Based Regular Expression Rule

By using the NEs already tagged in this sentence, the system creates a number of regular expressions, substituting suitable NE types into the ANSWER and OBJECT locations. Given the sentence: `John Lennon died on December 8th, 1980 during a public dramatic interpretation of J.D. Salinger's "Catcher in the Rye"`, the QITEKAT system would tag 1 DATE entity (December 8th, 1980) and 2 PERSON entities (John Lennon and J.D. Salinger) the QITEKAT system would dynamically produce 2 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 sentence 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 *A*).

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

Further examples of extracted *Knows* relations:

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

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

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

```
KnowsAbout(A, "Domain: PEOPLE",
   "George W. Bush", {K1, K2},
   1.0).
KnowsAbout(A, "Domain: PEOPLE",
   "John Lennon", Ka, 1.0).
```

A small number of broad domain types are used (PEOPLE, GEOGRAPHY, HISTORY, SPORT, MISC), and all relations are stored within the Knowledgeable Agents using serialized vectors, in order to achieve persistent data storage between executions.

B.2.3.2 Distribution

In developing the QITEKAT system, consideration was given to its use as a prototype for a Knowledge Grid (Cannataro and Talia, 2003), and for knowledgeable agents to communicate effectively with one another. This concept pointed toward the need for some kind of distributed system where agents could show mobility, and the ability to reside on a network, wherever there was data to process.

In addition, the large amount of data that was being handled for the Q&A task (1 million+ documents) lent itself to exploiting distributed paradigms to share the workload of examining this data and extracting suitable relations.

The QITEKAT system uses a simple UDP based system to handle communication between agents. This allows each agent to determine what other resources are available on the grid, and also inform others about the knowledge it possesses. As more agents are added to the grid, each becomes aware of what knowledge resources are available, and where a certain domain of questions may be best answered.

The system handles 5 message types:

Ping	Ask an agent if they are active.
Broadcast	Inform other agents in the grid that this agent is active.
Send_Question	Post a question to a specific agent on the grid.
Send_Answer	Send an answer back.
Send_KnowsAbout	Tell another agent what this agent has information about.

This approach allows knowledge to propagate through the system, as each question is sent from agent to agent to discover answers. When an answer is found, the response is returned, and the agents in the chain are each able to 'learn' that fact. A user only needs to enquire of a single agent in the grid, and that agent will be able to find the other agents on the grid that may be capable of answering the users query, and forward the question as required. A typical interaction between Knowledgeable Agents on this grid system is outlined below:

Agent 1 starts up, loads Knows and KnowsAbout relations and Agent IPs and sends a broadcast message on the local network.

Receives responses from other agents and updates its KnowsAbout relations.

Receives question from user.

Checks its own Knows relations for a suitable answer – none found.

Checks its KnowsAbout relations for another agent that may have an answer – one found (**Agent 2**).

Tags the question and forwards it to **Agent 2**.

Agent 2 finds an answer to the question and sends it back to **Agent 1**

Agent 1 updates it's knows relations so it now knows the answer and won't need to ask **Agent 2** next time.

Agent 1 forwards the answer to the user, and updates its local disk storage.

Figure B.5 – Typical Agent Interaction

B.2.4 Confidence Ranking

In the specific area of question answering it is often the case that systems are able to generate a number of candidate answers for a particular query. In this year's TREC Q&A track for example, an entire section of questions is devoted to returning multiple results for a single query (the so called List questions).

This poses the problem of determining the *best* result for a particular query, which is what is required by the standard questions in the Q&A track, and is likely to be the requirements of any practical application of a Question-Answering system.

The way in which this is often achieved is through a confidence ranking for an answer, reflecting the degree of certainty the system places on the answer returned being correct. The confidence ranking is often returned as a decimal value in the range 0.0 (zero confidence that the answer is correct) to 1.0 (completely confident that the answer is correct).

Past Q&A systems have used a number of means for determining a confidence measure from answers. Weighting based on matching NE types from the answer to that expected by a specific question type (i.e. A *where* type question expects a LOCATION type answer, and so a corresponding answer gets a higher weighting) is popular. Other popular measures include keyword densities in the answer document, and vector matching of question and answer pairs.

We adopted a new approach based on corroboration with external data sources (popular search engines)

B.2.4.1 Search Engine Corroboration

Search engines provide a large document base – Google for example currently claims to index over 3.3 billion Web pages, and as a result are likely to contain many examples of the correct answer to any query likely to be posed to a Q&A system. Although this offers scope to use Web search results as a source corpora for practical Q&A applications, the TREC Tracks require that all answers are found in the AQUAINT document collection. This does not preclude, however, the use of web search results to aid in the Q&A process, and we have adopted a novel approach for confidence ranking of answers, based on the results of an appropriate Web search query.

The fact that a suitable query to a search engine, based on the original question, is likely to result in many examples of the correct answer means that we can use the proportion of each possible answer within these search results to determine a relevance rank for that answer.

The QITEKAT system achieves this through a simple search API, developed in Java, which queries a number of popular search engines. Noun and verb phrase chunks from the question text are used to form a suitable search query, and the abstracts of the first 1000 results are retrieved from the search engine. These results are then scanned to determine the frequency of each of the possible results as produced by the Q&A system. The proportion of these frequencies are then used to calculate a relevance ranking.

This is better explained using a simple example:

- Given the question:
  When did John Lennon die?
- We extract the noun and verb phrases
  John Lennon
  Die
- These are then passed as a search query to Google
  "John Lennon" + "die"
- The first 1000 abstracts are retrieved
- The Knowledgeable Agents return three possible answers
  8th December
  15th August
  19th July
- Thus we find frequency matches for each of these answers in the Google abstracts, and calculate a relevance rating:

ANSWER	FREQ	CALC	RELEVANCE
8th December	462	462/533	**0.87**
15th August	28	28/533	**0.05**
19th July	43	43/533	**0.08**

Table B.1 – Relevance Ranking Calculation

So we have a corroborated relevance for each of the answers, and the Q&A system is able to return the answer 8th December as the most favourable.

B.3. Results

Preliminary testing of the QITEKAT system showed positive results on previous TREC question sets, and these are confirmed by the TREC 2003 evaluations.

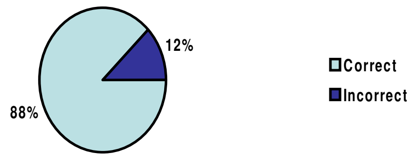
B.3.1 Trained Question Types

In developing the regular expression rules to extract *Knows* relations from source corpora we used the question data supplied as part of the TREC 2001 Question-Answering track. We constructed 400 regular expression rules, although time constraints meant we were unable to construct rules for all question types.

B.3.2 TREC 2002

Initial testing of the QITEKAT system was carried out on TREC 2002 Q&A Track questions in order to provide an indication of how the system would perform under typical application. Manual examination shows that of the 500 questions provided, rules have been constructed that should be able to find answers to 122 of them, assuming those answers exist within the AQUAINT source documents.

The system registered 107 correct answers, of which 4 were NIL answer questions, as no answer existed in the AQUAINT corpus. 15 incorrect answers were registered, of which 2 should have been NIL answers.



.

Figure B.6 - Results on TREC 2002 Questions

B.3.3 TREC 2003

Manual evaluation of the TREC 2003 question set showed that the system should have been able to answer 124 of the 500 questions made available with its current regular expression definitions. Evaluation of the TREC 2003 run showed 107 completely correct answers and 6 answers judged as being inexact. 11 incorrect answers were registered, which included answers that were judged as being unsupported answers.

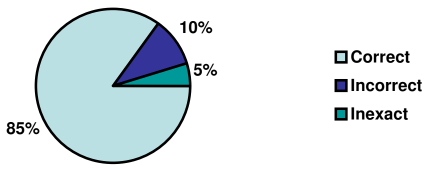


Figure B.7 - Results on TREC 2003 Questions

B.4. Analysis

The results produced by the QITEKAT system, both from in-house tests on previous Q&A data, and on the current TREC Q&A track questions are promising, particularly given the timescale of the development process. With levels of correct answers exceeding 80% in both tests, this implies a positive first step on the Q&A ladder, and a solid foundation to build on the work in Knowledgeable Agents and the concepts of '*Knowing About Knowledge*'.

B.4.1 Question Types

The results gained by the QITEKAT system need to be considered in the context of the question types that were addressed in order to gain a more accurate indication of the performance of the system.

It could be argued that the *When* and *Who* question types are the simpler of the main types used in the TREC evaluations, offering a definite answer type, and often more simple sentence constructs where an answer may be found. We felt this to be the case in this respect, and deliberately chose these types in order to aid the speed of system development in order to meet the deadline for run submission. We hope, however, that the underlying concepts of the system that we have adopted should be able to achieve similar results on all of the major question types, given suitable Regular Expressions on which to match.

B.4.2 Speed

Analysis of the AQUAINT documents which formed the source corpora for the TREC 2003 Q&A evaluation demonstrated the benefits of the distributed design adopted as the basis for the QITEKAT system, but also indicated a need for further speed improvements.

The final evaluation was carried out using a distributed network of 8 Pentium III computers, each using a 128Mb of local memory, and approximately 500Mb of local storage. The parsing and analysis of the 1 million documents took approximately 72 hours on this configuration. Although this level of performance is manageable, it would need to be improved if the system were to be applied to practical applications, or larger corpora, such as Web search results.

B.5. Future Directions

As a foundation for future Q&A and language processing research, the QITEKAT system has performed well, although a number of areas have been targeted as areas for improvement. In particular it is important that we are able to handle a greater number of question types in order to perform a more accurate evaluation of the systems performance, and allow for a direct comparison to other research systems participating in the TREC tracks. Further additional features that we feel may improve system performance, both in terms of speed of execution, and the ability to determine answers are outlined below.

B.5.1 Improved NE Classification

Although the NE classifier developed as part of the QITEKAT system performs well, for the purposes of Q&A 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 NE classifications (Sekine et al, 2002), which we feel would be a beneficial addition to the QITEKAT architecture.

B.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 (Miller, 1990) 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 QITEKAT 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.

B.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 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.

B.5.4 Automate RE Production

Manual production of Regular Expressions to extract information from document texts was one of the more time consuming aspects of the initial QITEKAT development, and as a result meant we were only able to focus the tool at a limited number if question types in order to meet the TREC deadline. A key idea for future development of the system is to implement an automated system, capable of producing generic expressions which could then be used to extract further information. Initial thoughts are that this issue may lend itself to a transformation based system, similar to that found in Brill-type POS tagging systems (Brill, 1992), where it would be possible for the system to learn a set of rules, based on existing, manually produced REs.

B.6. References

E. Brill. 1992. "A simple rule-based part-of-speech tagger" In *Proceedings of ANLP-92*, pp 152–155, 1992.

M. Cannataro and D. Talia. 2003. "The Knowledge Grid". In *Communications of the ACM* Vol 46, Number 1, pp 89-93.

J. Chu-Carroll, J. Prager, C. Welty, K. Czuba and D. Ferrucci. 2002. "A Multi-Strategy and Multi-Source Approach to Question Answering". In *TREC 2002 Proceedings*.

J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. 1996. "Fastus: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text" In *Finite State Devices for Natural Language Processing*. MIT Press, Cambridge MA.

G. Miller. 1990. "WordNet: An On-Line Lexical Database" In *International Journal of Lexicography.*

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

W. Teahan and D. Harper. 2003. "Using Compression-Based Language Models for Text Categorization" *Language Modelling for Information Retrieval,* Chapter 7, pp 141-165. Kluwer Academic Publishers.

W. Teahan. 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.

Generating GeneRIFs for a Multi-Agent-based Biomedical Information Retrieval System

Abstract

This section describes work that was done for the 2003 Text Retrieval Conference (TREC) Genomics Track second task. It also describes preliminary work on implementing a multi-agent Biomedical information retrieval system. The proposed multi-agent system will apply knowledgeable agents using a logic-based question and answering framework. Part of this work requires the specifying of context for the questions and answers, and one means of doing this is to generate GeneRIFs for each biomedical document, where a GeneRIF is a MEDLINE standard for describing the contents of the biomedical document in terms of gene function. Various methods are explored to generate the GeneRifs automatically.

C.1 Introduction

The Text Retrieval Conference (TREC) Genomics Track was started to provide a forum for information retrieval in the genomics area. The Secondary Task for this year's track is to analyse automatic methods to reproduce the GeneRIF notation for biomedical papers. Below shows the official definition of what a GeneRIF should be:

A concise phrase describing a function or functions (less than 255 characters in length, preferably more than a restatement of the title of the paper.)
(www.ncbi.nlm.nih.gov/LocusLink/GeneRIFhelp.html)

The GeneRIF annotation is designed to allow the contents of a bioscience paper to be summarised in such a way as to show the function of the gene, which is the subject of the bioscience paper. An analysis by Mork and Aronson (2003) of NLM found that 95% of GeneRIF snippets contained some text from the title or abstract of the article. About 42% of the matches were taken directly from the title or abstract, 25% contained significant runs of words from pieces of the title or abstract.

The data provided for the Secondary Task consists of 139 GeneRIFs representing all of the articles appearing in five journals – Journal of Biological Chemistry, Journal of Cell Biology, Nucleic Acids Research, Proceedings of the National Academy of Sciences, and Science – during the latter half of 2002.

C.2 Methodology

The main objective of our research is to design and implement a multi-agent based system for knowledge-based retrieval to biomedical literature. The purpose is to provide easy-to-use and seamless interfaces to biomedical literature both for the expert and for the layperson. The main components of the multi-agent systems we envisage will consist of peer-to-peer based communicating agents which are knowledgeable about biomedical resources. The biomedical information retrieval systems will provide scientists in the biomedical community with better support for searching the latest literature, therefore enabling them to be better informed about the latest developments in the biomedical field. The systems will also serve the wider community that will enable researchers, whether they are experts or non-specialists, a seamless and natural interface that will require minimal training.

C.3 Knowledgeable Agent Framework

We are in the process of designing and developing a novel logic-based framework for implementing knowledgeable agents that will become the core components for our multi-agent biomedical information retrieval system. The logic is based on three relations that describe whether an agent is "knowledgeable" or not (Teahan, 2003; also see the first part of this paper).

C.3.1 Context modelling

We feel that context has a very important role to play in specifying knowledge. For example, for an agent to answer the question "What is entropy?"' in a knowledgeable way, the agent must first appreciate the context in which the question is asked. A different answer is required to this question depending on whether the context concerns the domain of physics or the domain of information theory. If neither domain is apparent given the context, then it might be appropriate for an agent (if it wishes to be knowledgeable) to inform the person asking the question that more than one answer is possible to the question. We feel that different representations of context are required in different applications depending on the nature of the knowledge that needs to be specified and/or manipulated.

C.3.2 GeneRIF as a context

For the application that we investigate in this paper, we explore the possibility of using GeneRIF-based annotation for specifying the context of biomedical documents in the Genomics domain. Importantly, in this "context", we consider the MEDLINE GeneRIFs as provided in the training data for the TREC 2003 Genomics Secondary Task experiments as only a representation of the "true"' GeneRIF. By "true", we mean what a correct annotation of the article might be if it was performed by a group of experts, rather than the GeneRIFs encountered in the MEDLINE database. Note that some of the MEDLINE GeneRIFs in the training data evidently fall short of what a group of experts might assign to the "correct" or "true" GeneRIF, if they had been allocated this task.

In light of this, we can also partially ignore the differences between the MEDLINE provided GeneRIF and the candidate GeneRIFs we are automatically generating, since we are only interested in finding a GeneRIF description that encompasses some notion of a true GeneRIF. Hence, providing GeneRIF strings that are potentially longer than the MEDLINE GeneRIF, but have a much greater chance of encompassing the "true" GeneRIF description makes sense, as our goal is to ensure as comprehensive description as possible is generated for information retrieval purposes. Our contention is this will make it more likely that a relevant document is retrieved when the GeneRIF is used in an IR system – in our case, within the distributed IR framework that we are developing based on Knowledgeable Agents.

The consequence of this is that we feel that a modified co-efficient is more suitable for our purposes to evaluate the "goodness" of the generated GeneRIF. This modified co-efficient is a measure of the percentage of words in *X* that occur in *Y*, so we are effectively ignoring the extraneous words in *Y*. The equation below is for the modified measure:

$$D'_{(A,B)} = Z/X \qquad [1]$$

as opposed to the original DICE co-efficient:

$$D'_{(A,B)} = 2 \times Z/(X+Y) \qquad [2]$$

where *A* is the MEDLINE GeneRIF, *B* is the generated GeneRIF, *Z* is the number of words that occur in both *A* and *B*, *X* is the number of words in *A*, and *Y* is the number of words in *B*.

Changing the DICE co-efficient in this way shows that we are only interested in the following questions: "Does the generated GeneRIF contain the MEDLINE GeneRIF?" or "How much of it does it contain?" These questions are noticeably different to what the standard, unmodified DICE co-efficient measures which is: "How similar are the MEDLINE and generated GeneRIFs?"

C.3.3 Experimental Results

Table C.1 shows the results for the secondary task experiments, using both the standard DICE co-efficients (Classic DICE (see equation [2]), Modified Unigram DICE, Modified Bigram DICE, and Modified Bigram Phrases DICE), and the modified measure (see equation [1]). The table is divided into sections by experimental run, and these sections are divided further by the co-efficient used to generate the average result for the particular run. The runs processed the following data: *uwb2* used a concatenation of the document titles and the last line of the document abstracts as input data, *uwb3* used the document titles as input data, *uwb4* used the last line of the document abstracts as input data, and *clairvoyant* used pre-calculated DICE co-efficients to find the best possible generated GeneRIF from a choice of document title, first line of document abstract, penultimate line of the document abstract, and last line of document abstract. The *clairvoyant* run is therefore a measure of the best possible result that the chosen generated GeneRIF selection technique could produce, but used data that would not be available to the system when it operates autonomously. Due to this, only runs *uwb2*, *uwb3*, and *uwb4* were submitted to TREC for evaluation. The use of the modified measure in the *clairvoyant* run is justified as the run produces the best standard DICE co-efficient results and the second best modified measure result.

The result the modified measure achieved (61.48%) indicates the better coverage of the input data of run *uwb2*. This is unsurprising, as the input data for this run was a concatenation of two strings; so the modified measure had more data to work with. However, incorrect data for run *uwb2* was sent to TREC, but this did not have an effect on the results for the classic DICE, the modified unigram DICE, or the modified measure in this run. Our own analysis of the corrected data gives the results shown in table C.2.

The best run using the standard DICE co-efficients was the Modified Unigram DICE co-efficient in run *uwb3*, which produced 48.25%. This result is interesting as it shows that document titles do contain pertinent data for generating GeneRIFs. However, this advantage is lost when document titles are combined with other parts of the document, such as the last line of the document abstracts in run *uwb2*.

Table C.1: Experimental Results

CO-EFFICIENT TYPE	Average %
uwb2	
Classic DICE	44.41
Modified Unigram DICE	44.07
Modified Bigram DICE	2.33
Modified Bigram Phrases DICE	1.80
Modified Measure	61.48
uwb3	
Classic DICE	46.48
Modified Unigram DICE	48.25
Modified Bigram DICE	29.53
Modified Bigram Phrases DICE	32.82
Modified Measure	42.74
uwb4	
Classic DICE	36.28
Modified Unigram DICE	35.21
Modified Bigram DICE	22.73
Modified Bigram Phrases DICE	24.52
Modified Measure	38.80
clairvoyant	
Classic DICE	58.16
Modified Unigram DICE	59.61
Modified Bigram DICE	45.04
Modified Bigram Phrases DICE	47.94
Modified Measure	54.43

Table C.2: Corrected Results for *uwb2*

CO-EFFICIENT TYPE	Average %
uwb2	
Classic DICE	44.53
Modified Unigram DICE	40.08
Modified Bigram DICE	29.20
Modified Bigram Phrases DICE	31.80
Modified Measure	-

C.4 References

J. Mork and L. Aronson. July 2003. medir.ohsu.edu/~genomics/protocol.html.

W. Teahan, 2003. "Knowing about knowledge: Towards a framework for knowledgeable agents and Knowledge Grids", Tech Report AIIA 03.2, School of Informatics, University of Wales, Bangor.