Natural Language Question Answering Over Triple Knowledge Bases

Aaron Defazio
Supervisor: Scott Sanner

October 30, 2009

Abstract

Question answering systems have been the focus of widespread research. They have the potential to make finding information even easier than with modern search engines. Successful question answering systems have to overcome several hurdles. A knowledge representation language has to be chosen that is flexible to represent the majority of information in the source corpus, while still allowing for tractable extraction and inference. A robust method for translating free-form sentences from the source corpus to the knowledge representation language is needed, along with a method for translating questions into queries over the knowledge base.

This report describes a question answering system that uses triples (subject-predicate-object 3-tuples) as a knowledge representation language. A method for extracting triples from paragraphs of text using a series of simplification passes is described. A basic logic based query language is also described, along with a method for translating these queries to efficient SQL queries over a database. An accompanying method for translating questions using a top-down search of subsets of this query space is detailed. The method makes use of only shallow syntactic processing and is notable as a tractable application of an exhaustive, uninformed search.

Tests of the system on a small scale are shown with promising results, 35% of questions being answered by one of the top 5 returned answers.

Contents

Introduction

Natural language question answering is a discipline of machine learning concerned with the direct answering of questions posed in natural language. This report will focus on questions asked in English. Most people are familiar with document retrieval systems of some kind. These have come into widespread usage; web search engines being the primary example. Typical document retrieval systems perform little to no natural language processing of the question, often performing keyword like matches. In comparison, a question answering system returns an exact answer to the users query, typically a single word or phrase. A successful question answering system reduces the time and effort required for the user to find their answer. However, question answering systems are still an area of active research, and even tasked with the problem of extracting a 250 character snippet of text containing the answer (a much simpler problem), current state of the art systems can only answer 70% of posed questions (Dang, Kelly, and Lin, 2007).

This report describes a question answering system built over a triple knowledge base. Typical document retrieval systems and some question answering systems store information as plain text, as this best facilities keyword searches. An alternate approach is to store the corpus in a processed form, one which attempts to attach semantics to fragments of text to enable more powerful retrieval mechanisms. The system described in this report uses *triples* as the knowledge representation language. Triples (some times referred to as triplets) are 3-tuples of information, consisting of a subject, a predicate and an object. The intended interpretation is that the subject is acting on the object in a way described by the predicate. Typical triplets have noun phrases as the subject and object and verb phrases as the predicate, although often adjectives make meaningful objects, usually when the predicate is the copula *be*. Some examples of triples are:

(amazon rainforest , be , forest)
(appalachian trail , traverse , maine)
(eurostar train , get , 22000 passengers each day)

In practical triple extraction systems parentheses and commas can not be part of triples, so no quoting of the triple components is necessary. Text is usually normalized including a reduction to lowercase, as is the case in these examples.

In the worst case, answering a question requires deep background knowledge and reasoning capabilities outside the reach of current natural language processing technologies. This can occur in questions whose answers are equivocal in nature, or in questions which can not be adequately be answered with a single phrase. A typical example of such is:

In what country did the game of croquet originate?
Attempting to answer this question from information extracted from the Wikipedia page for croquet would be difficult as several sentences suggest possible conflicting answers:

The evidence suggests that the rules of the modern game of croquet arrived from Ireland ...
The first explanation is that the ancestral game was introduced to Britain from France ...

Some questions which seem reasonably straight forward actually require some amount of deduction to be answered. Consider the following question:

How old is Kevin Rudd's wife?

Answering this question from a text such as the following requires the combining of two separate facts, as there is no direct answer.

Therese Rein (born 17 July 1958) is an Australian businesswoman and the wife of the 26th Prime Minister of Australia, Kevin Rudd.

The above sentence contains 5 recognizable facts, of which 2 are required to answer this question. This sort of density of information in sentences is normal for encyclopedic text. This also highlights the complexities of converting a text corpus into a usable knowledge base. Natural language is often ambiguous, and can require context outside what can be gleaned from a sentence by sentence window.

Because of these difficulties, the system described in this report is focused on answering questions which have unequivocal, single phrase or word answers. This sort of question is referred to as a *factoid* question. Questions of this type are widely used for comparison & evaluation of question answering systems (Voorhees and Harman, 2000).

Background

In this section, the natural language processing technologies that were used are described, along with two ontologies that were used for query expansion and triple simplification. The triple representation is made precise, and the differences from existing triple representations are discussed. This provides the foundation for which the triple extraction heuristics build on. A logical query language for querying triple knowledge bases is defined, with structure in the form of a partial ordering relation, so statements about queries being more or less general then each other are well defined. This will allow the formulation of a top down search strategy in the next section, in which the children of a node are all less general then their parent.

The Language Pipeline

A Triple based question answering system needs to preprocess its corpus into a triple knowledge base. This involves the application of a variety of natural language processing techniques. To achieve this, most natural language processing systems are built around a processing pipeline design. Typically documents start as plain text, and are feed through the stages of the pipeline, with each stage preforming some kind of syntactic or semantic analysis. Most of the stages are common between systems, and high quality libraries exist to perform these functions. The following are the processing stages used in this system for processing of an input corpus before triple extraction starts:

1. **Sentence Extraction** involves finding which of the sentence delimiting punctuation characters are being used to separate sentences, and which are being used for other purposes such as ending abbreviations. Sophisticated sentence extractors also handle sentence punctuation occurring inside of quotations, and other corner cases. The processing of a few sentences will be shown to illustrate this and other steps of the language pipeline:

Malcolm X (born Malcolm Little; May 19, 1925 – February 21, 1965) was an African-American Muslim minister, public speaker, and human rights activist. To his admirers, he was a courageous advocate for the rights of African Americans.

Following sentence extraction:

1) Malcolm X (born Malcolm Little; May 19, 1925 – February 21, 1965) was an African-American Muslim minister, public speaker, and human rights activist.
2)To his admirers, he was a courageous advocate for the rights of African Americans.

2. **Tokenisation & Part Of Speech Tagging Of Words.** Part of speech tagging is the task of identifying the grammatical tag of each word. The notation usually used for this involves placing the POS tag after each word, separated by a forward slash. The most common POS tags are included as part of Table 1. The above sentences with POS tagging are:

1) **Malcolm**/NNP **X**/NNP **-LRB-**/-LRB- **born**/VBN **Malcolm**/NNP **Little**/NNP ;/: May/NNP **19**/CD **,**/, **1925**/CD **–**/: **February**/NNP **21**/CD ,/, **1965**/CD **-RRB-**/-RRB- **was**/VBD **an**/DT **African-America**n/NNP **Muslim**/NNP **minister**/NN **,**/, **public**/JJ **speaker**/NN **,**/, **and**/CC **human**/JJ **rights**/NNS **activist**/NN **.**/.
2) **To**/TO **his**/PRP$ **admirers**/NNS **,**/, **he**/PRP **was**/VBD **a**/DT **courageous**/JJ **advocate**/NN **for**/IN **the**/DT **rights**/NNS **of**/IN **African**/NNP **Americans**/NNPS **.**/.

3. **Named Entity Recognition** is a pass that identifies blocks of sequential words that form a entity. Entities are locations, persons, organizations, times, quantities, monetary values, percentages and the like. Most named entity systems also tag each entity with its type, although that information is not used in the system described here. The entities identified in the example text are:

Malcolm X; Malcolm Little; May 19, 1925; February 21, 1965; African-American Muslim minister; public speaker; human rights activist; his; he; courageous advocate; the rights; African Americans.

4. **Coreference Resolution** attempts to identify the target of words that reference entities in a sentence. Common instances of references include personal pronouns (e.g. *he, she, it*), nick names or short names (e.g. *The president*, *Obama, Barack Obama* all refer to the same person at the time of writing), or more general anaphoric noun phrases. The reference *the country*, in the context of an article about Australia, would be an example of this more general case. By far the most common case is that of personal pronouns, identified with the POS tag PRP (see Table 1). The system described in this report just replaces these

references with the target entity, although a more robust system would keep the original coreference as well. The second sentence translates to the following under basic anaphora resolution (Note that no attempt to get correct plurality of the inserted entity is made, as plurality distinctions are removed in a later stage):

To Malcolm X admirers, Malcolm X was a courageous advocate for the rights of African Americans.

5. **Full Parsing** is the forming of parse trees showing attachments and phrasal structure. The full parse tree for the first of the example sentences is:

```
(ROOT
  (S
    (NP
      (NP (NNP Malcolm) (NNP X))
      (PRN (-LRB- -LRB-)
        (VP (VBN born)
          (NP
            (NP (NNP Malcolm) (NNP Little))
            (: ;)
            (NP (NNP May) (CD 19) (, ,) (CD 1925))
            (: --))
          (NP (NNP February) (CD 21) (, ,) (CD 1965)))
        (-RRB- -RRB-)))
    (VP (VBD was)
      (NP
        (NP (DT an) (NNP African-American) (NNP Muslim) (NN minister))
        (, ,)
        (NP (JJ public) (NN speaker))
        (, ,)
        (CC and)
        (NP (JJ human) (NNS rights) (NN activist))))
    (. .)))
```

   This parse tree is written in Penn Treebank (Bies, Ferguson, Katz, Macintyre, Tredinnick, Kim, Marcinkiewicz, and Schasberger, 1995) style, using a lisp like notation. The tags uses are explained in Table 1. Nesting level is denoted with tabs. The basic NP-VP phrase structure is evident, nested beneath S at the same level as the full stop. Named entities typically appear as noun phrases in parse trees if correctly parsed.

The system described in this report uses the Stanford parser (Klein and Manning, 2003) for 1, 2 and 5 and the BART coreference toolkit (Versley, Ponzetto, Poesio, Eidelman, Jern, Smith, Yang, and Moschitti, 2008) for 3 and 4. The triple extractor works directly on the parse trees output from step 5.

tag	meaning
ADJP	Adjective Phrase
ADVP	Adverb Phrase
CC	Coordinating conjunction
CD	Number
DT	Determiner
FRAG	Fragment
IN	Preposition or subordinating conjunction
JJ	Adjective
LRB	Left Parenthesis
NN	Noun, singular or mass
NNP	Proper noun, singular
NNPS	Proper noun, plural
NNS	Noun, plural
NP	Noun Phrase
PP	Prepositional Phrase
PRN	Parenthetical
PRP	Personal pronoun
RRB	Right Parenthesis
S	simple declarative clause
SBAR	Clause introduced by a subordinating conjunction
SBARQ	Direct question
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
VP	Verb Phrase
WDT	Wh-determiner
WHADJP	Wh-adjective Phrase
WHAVP	Wh-adverb Phrase
WHNP	Wh-noun Phrase
WHPP	Wh-prepositional Phrase
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table 1: Penn Treebank tags used by Stanford parser (Bies et al., 1995).

Ontological Sources

Wordnet

Wordnet (Fellbaum, 1998) is a lexical database of English, cataloging the inter-dependencies between words. It can be of use in the post processing of triples, as its database of morphological roots can be used to bring words into a canonical form. For example, the following two triples have the same semantics, but this is not captured by the triple representation until each word is replaced with its morphological root.

(John , was born, 14 July)
(John, be bear, 14 July)

In this case *was* is the past tense of the bare infinitive *be*, and similarly *born* is the past tense of the bare infinitive *bear*.

Words that Wordnet can not identify the roots for are left as is. Bringing words to their morphological root removes tense and plurality distinctions, reducing words to their infinitive form when possible. Using morphological roots in this way, or other semantically aware word simplification techniques is known as *lemmatisation*, as opposed to *stemming* (Lovins, 1968), an approach which just simplifies words based on their structure (i.e. removal of prefixes and suffixes). A stemmer would reduce all of the following words to operate (Manning, Raghaven, and Schutze, 2008):

operate operating operates operation operative operatives operational

Stemming would not be able to reduce the above triples to the same form however. Stemming often gives better results for search term processing in an information retrieval context (Hollink, Kamps, Monz, and de Rijke, 2003). However, it was found in the context of this triple processing task that stemming would result in less human readable triples. Stemming produced stems which were not valid words more often than lemmatisation did.

Wordnet is also used for query expansion. This involves replacing query words with a set of synonymous words in the hopes that it will increase recall. For example, Wordnet finds for the word *surname* the synonyms *last name* and *family name*. An expanded query using those synonyms would return better results, as all three forms are in common use. The only downside with using synonym sets (synsets) in this way is that a synset is associated with a words meaning, and for words that can have several meanings it is not clear which of the associated synsets should be used. Using all associated synsets would increase recall at the expense of accuracy. The system described in this report uses the synset associated with the most common meaning of a word.

Wikipedia Redirects

While lemmatisation is effective for most language constructs, other techniques have to be used for canonicalising entities such as names, places, times and dates. During the coreference resolution stage of the language pipeline, it is useful to replace all occurrences that are found to be coreferential with a canonical form of the referred to entity. Recent research has found Wikipedia structure and text to be a useful source of ontologies (Syed, Finin, and Joshi, 2008, Nakayama, 2008). One such resource available is the Wikipedia redirect data. Wikipedia contains a large number of redirects, which send a user who enters a search term to a page that covers the entered search term. These each redirect can be treated as a synonym or hypernym relation. If there is a redirect from phrase a to b, then a is a synonym or hypernym of b. Furthermore b is typically the most canonical form of the entity referred to (This is Wikipedia policy, see wik). Some examples are:

duluth hs → duluth high school
ramer → ramer, tennessee
the dirty knobs → mike campbell
marius cristian negrea → marius negrea
volvo 142 s → volvo 140 series
bucheon fc → bucheon sk
monacacy battlefield → monocacy national battlefield
saypan → saipan

One of these examples also illustrates a pitfall of this technique. *The dirty knobs* is not a direct reference to *mike campbell*, but rather a band which he is a part of (a meronym). Wikipedia redirects are sometimes used to link a narrow topic to a article for a more general topic which also discusses it, usually if the narrow topic doesn't have its down Wikipedia page.

Triples with Modifiers

As described before, triples are 3-tuples of subject-predicate-object information. This is a simplification however, as realistic triple knowledge bases need to store modifiers (or adjuncts) of the three components. The most common approach is to store both word and phrase modifiers ordered as attachments to the part of the triple it is modifying, including sub-attachments. An example of this is:

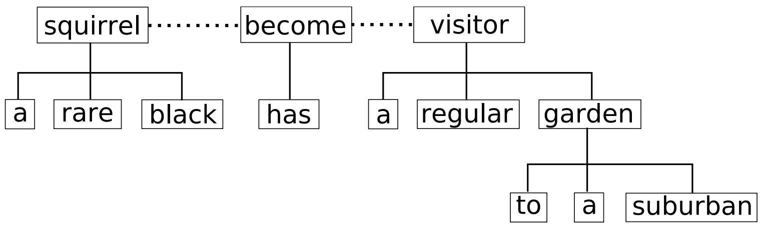

Figure 1: Triple from Rusu, Dali, Fortuna, Grobelnik, and Mladenic, 2007.

The problem with this additional modifier structure is that it becomes more difficult to search over than a simple triple, and it becomes more fragile with regard to misparsed attachments. Incorrect attachment of modifiers is still a common problem with modern parsers, so some level of robustness is desirable.

The approach used here has a coarser structure for attachments. Modifying words are included in the part of the triple they modify (i.e *rare*, *black* would be included in the subject of the triple above as *rare black squirrel*), whereas phrasal modifiers are handled differently. Phrasal modifier attachments are made to the triple, not to the parts of the triple. Furthermore, no nesting is allowed. Phrasal modifiers are attached in a n-1 relationship with triples, and can only modify triples as opposed to the more general case with triples and other modifiers. The same sentence as Figure 1 using this triple formulation is:

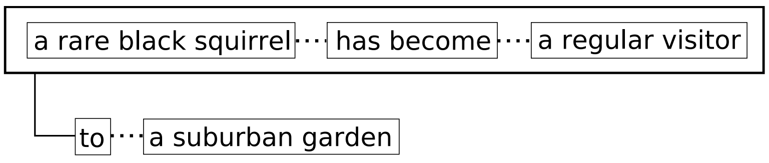

Figure 2: Triple form used by the system described in this report.

Notice that the head of the modifying phrase has been separated from the body of the phrase. Modifiers are stored this way so as to allow searching over the two parts separately. The words *a* and *has* in the above triple do not provide useful information, and are removed as stop-words during triple post processing.

A Simple Logical Query Language

This report uses a logical formulation of triple based knowledge. This allows the techniques and notation of logic to be used to describe the translation and evaluation of queries. Consider the triple form described above. We can represent this in the predicate calculus using two logical predicates, T and M (for triple and modifier respectively), where T takes 4 arguments (identifier, subject, predicate, object) and M takes 3 (identifier, head, phrase). The first component of T and M is the triple identifier used for showing the attachments. A modifier is *attached* to a triple if its first component is the triple's identifier. The form of the identifier is immaterial, but in practice integral numbers are used. The constants of this logical formulation are general strings. So using this notation for the triple in Figure 2, the following is a true statement given our knowledge base:

$$\begin{aligned}\exists t. \quad & T(t, \text{"a rare black squirrel"}, \text{"has become"}, \text{"a regular visitor"}) \\ & \wedge M(t, \text{"to"}, \text{"a suburban garden"})\end{aligned}$$

The identifier is existentially quantified as its actual value is not important, only that an identifier exists that links together the two logical predicates.

Querying over such a knowledge base is just finding all models that satisfy a given logical expression. To make query evaluation practical and translatable to queries on relational databases, it is convenient to only allow a subset of all possible logical expressions as queries. For the purpose of this report we define a *simple logical query* as one which:

- Consists of a disjunction of conjunctions of predicates. The disjunction is allowed to contain a single set of conjunctions.
- Each argument of the predicate is a String or variable.

Quantifiers are not necessary for this sort of querying, although they could be used as a form of projection if they were allowed. This is a more restricted version of disjunctive normal form where negation of variables is disallowed. Other than the quantifier, the logical statement above is a query in this form.

θ-subsumption and the Structure of the Simple Logic Query Space

In Inductive Logic Programming (ILP), a *hypothesis* space of logic programs is formulated in much the way the simple logic query space is defined above. Like in ILP, the space is purposefully restricted so as to make searching of the space possible. This restriction is known as the *language bias* (Lavrac and Dzeroski, 1994). To make top down searching of the simple logic query space feasible, a structure in the form of a partial ordering needs to be introduced. One such structure is the concept of θ-subsumption:

Substitution A *substitution* $\theta = \{x_1 = T_1, ..., x_k = T_k\}$ is a function from variables to sets of terms. The application $W\theta$ of a substitution θ to a logical expression W is a obtained by taking a disjunction of copies of W, formed by taking for each $(t_1, ..., t_k) \in T_1 \times ... \times T_k$ a copy with occurrences of each variable x_j replaced by t_j.

θ-subsumption Let c and c' be two clauses. Expression c θ-subsumes c' if there exists a substitution θsuch that $c\theta \subseteq c'$(i.e the top level clauses are a subset).

This is a more general definition of a substitution then what is usually used (For example in Lavrac and Dzeroski (1994)), as terms rather then term sets are usually used. For single element sets appearing in substitutions, the set notation will be omitted in this report for notational convenience.

The partial ordering this induces coincides with making queries more general (or conversely more specific). For instance, a clause c is at least a general as a clause c'if c θ-subsumes c', and it is more general if the converse is not true. In ILP the terminology *generalisation* and *specialisation* is used. A more general query will always return more results, and the results returned will be a super-set of those from the less general query. The system discussed in this report focuses on specialization.

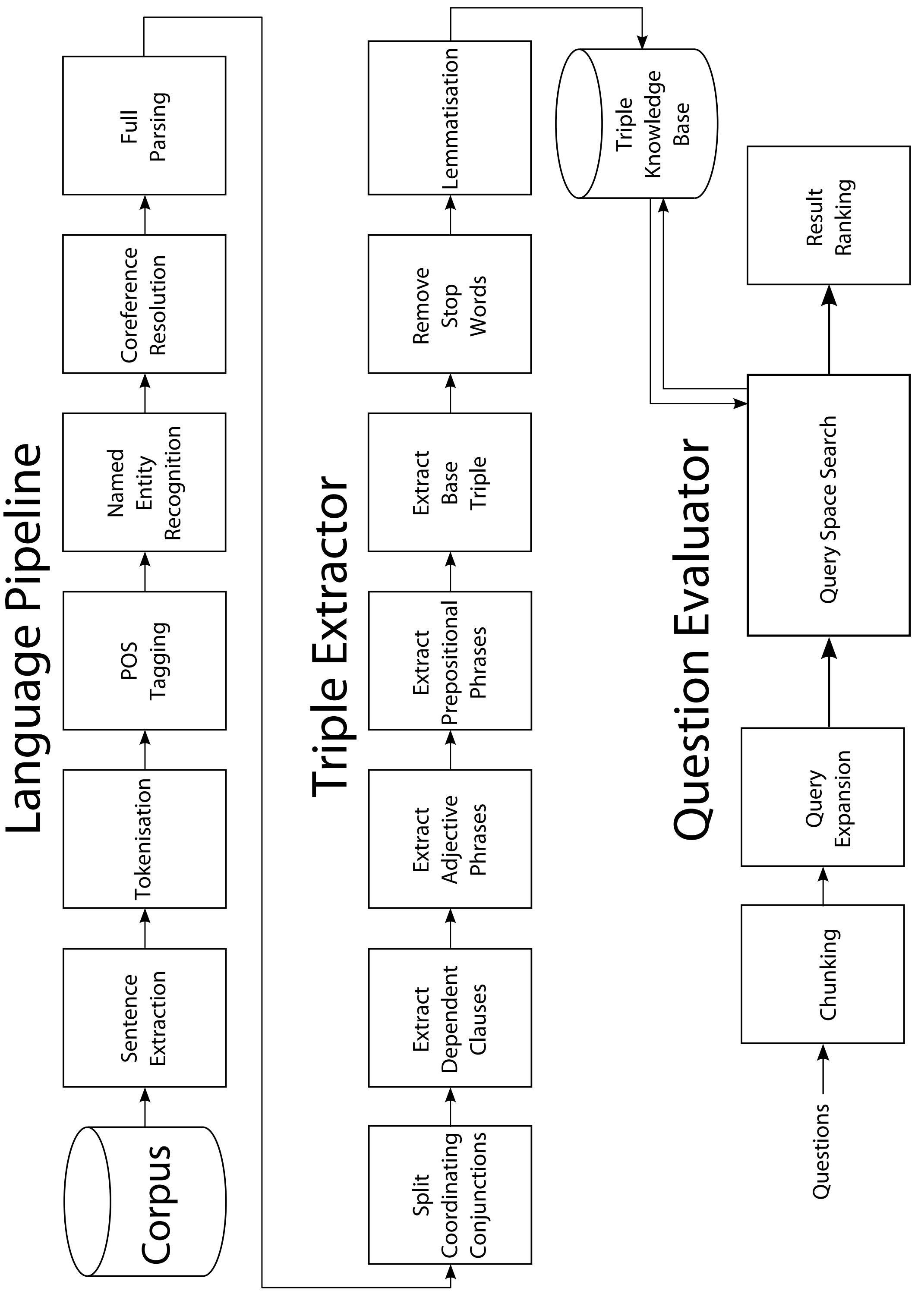

Figure 3: System architecture.

Approach

Triple Extraction

Triple extraction is done with a heuristic approach. Parse trees pass through a series of simplifications, each of which may split trees in some way or attach triple modifiers to a list of such modifiers carried as meta-data with a tree. At the end of the simplification passes there will be some number of trees, each of which is expected to follow the NP-VP-NP pattern (where either NP is optional). The modifiers that were attached to each tree are then attached to the associated triple. This process is detailed in Algorithm 1 (see appendix A for pseudo-code of the individual passes).

`SplitCoordinatingConjunctions`

The Split Coordinating Conjunctions step is designed to simplify sentences making heavy use of conjunctions. As long as the parse tree is nested correctly, it will break sentences expressing the same fact about a list of things, into several sentences, each about one fact. For example, it successfully splits the following:

William Henry Seward was a Governor of New York and the United States Secretary of State under Abraham Lincoln and Andrew Johnson.

into:
William Henry Seward was a Governor of New York under Abraham Lincoln.
William Henry Seward was a Governor of New York under Andrew Johnson.
William Henry Seward was the United States Secretary of State under Abraham Lincoln.
William Henry Seward was the United States Secretary of State under Andrew Johnson.

This sort of splitting can cause problems with dates, times, names or more general collocations which involve conjunctions. Part of the named entity extractor's function is to detect these cases, so if it is functioning correctly conjunctions inside of entities will not be split.

Algorithm 1 Triple extractor core. See appendix A for code for passes.

```
type TreeWithModifiers = Tree + List of triple modifers

val passes = [
    SplitCoordinatingConjunctions,
    ExtractDependentClauses,
    ExtractAdjectivePhrases,
    ExtractPrepositionalPhrases]

def (xs map f) = transform each element of list by applying f
def (xs flatMap f) = map then concat list of lists to single list

def extractTriples( t : TreeWithModifiers) = {
    var results = [t]

    for( pass <- passes ) {
        results = results flatMap pass
    }

    return results map extractSimpleTriple
}

def extractSimpleTriple( t : TreeWithModifiers ) = {
    var st = first NP or VP in a depth first search(DFS)
    if st is not NP
        st = null

    vpt = first VP in a DFS
    ot = first {NP,ADJP,JJ,JJR,JJS} under vpt in a DFS
    pt = vpt with ot removed if its not null

    Triple(st, pt, ot) + t's modifiers
}
```

`ExtractDependentClauses`

The Stanford parser identifies all clauses in a parse tree as one of S, SBARQ, FRAG, or SINV. This pass excises each clause from the main parse tree into a new parse tree, along with a clone of its subject phrase. Sentences with dependent clauses, known as *complex* sentences in linguistics—as opposed to *simple* sentences with a single clause—are common in encyclopedic text. A dependent clause is introduced by either a subordinate conjunction (for adverbial clauses) or a relative pronoun (for relative clauses), so those two cases have to be handled differently. This pass also extracts parenthesised phrases and clauses as they can be handled similarly, although not all are technically dependent clauses. Adverbial clauses are extracted into modifiers, whereas relative or parenthesised clauses are broken off into separate sentences.

Some examples of its use are:

Relative clause

Laughter erupted from Annamarie, who hiccuped for seven hours afterward.

into
Laughter erupted from Annamarie.
Annamarie hiccuped for seven hours afterward.

Parenthesised clause

Magna Carta (the Great Charter of Freedoms) is an English legal charter.

into
Magna Carta is an English legal charter.
Magna Carta be the Great Charter of Freedoms.

Adverbial Clause

John Smith died when he was young.

into
John Smith died.
with attached modifier
(when, he was young)

`ExtractAdjectivePhrases`

Adjective phrases typically appear in sentences between one or two commas, and appear in the parse tree as nested under their subject. They are common in encyclopedic text, and capturing them is an essential part of a robust triple extractor. This step transforms the following:

The Statue of Liberty, dedicated in October 1886, commemorates the centennial of the signing of the United States Declaration of Independence.

into
The Statue of Liberty dedicated in October 1886.
The Statue of Liberty commemorates the centennial of the signing of the United States Declaration of Independence.

`ExtractPrepositionalPhrases`

Prepositional Phrases are the main type of adjunct that is converted into a triple modifier. They are also easy to extract, as they appear as PP nodes in the parse tree. Because the attachments of modifiers are ignored by this system, attachments don't need to be captured.

An example is:

The average distance from Earth to the Moon is 384403 kilometers.

into
The average distance is 384403 kilometers.
with attached modifiers
(from, Earth)
(to, the Moon)

Question Evaluation

This system does not use a direct translation approach to question answering. Converting questions to a single logic query, or even a set of queries, is a fragile process. Instead, a search of a restricted space of possible queries is performed. This space can contain thousands of queries. While time consuming, this sort of method is potentially more robust. Differences in question phrasing, scope or structure from potential answers is handled in a way that requires only shallow syntactic analysis of questions.

Question Preprocessing

A question needs to be broken down into parts that are of the same sort of form as the parts of a triple. Triples often contain multiword entities like *Yiddish Policemen's Union*. If this entity occurred in a question then attempting to match triples with these individual words as the subject or object would fail unless a full text matching search was used. The system described in this report performs exact matching, so it is necessary to group the words of each entity together. The OpenNLP chunk parser performs this function. A chunk parser (Abney, 1991) is a way to break sentences up into conceptual units. Chunks typically contain one content word or a single entity, plus its consecutive function and modifier words. The idea is that the scope of each chunk produced tends to match the content of parts of triples.

The chunks are then classified based off of the part of speech tag given to the last word in the chunk. Noun or adjective chunks are given the class *entity* and all others are classified as *assoc*. The idea behind these classes is that entity chunks can be unified with the subject or object of a predicate or the body of a modifier, whereas assoc chunks can be unified with triple predicates or the head of modifiers. Chunks are then processed by removing stop-words in entity chunks and each chunk is converted to lowercase and lemmatised. Some example chunked questions are:

Question: How long does it take to travel from London to Paris via the Chunnel?
[HOW / ASSOC, TAKE TO TRAVEL / ASSOC, FROM / ASSOC, LONDON / ENTITY, TO / ASSOC, PARIS / ENTITY, VIA / ASSOC,CHUNNEL / ENTITY]
Question: Who created the board game Pictionary?
[CREATE / ASSOC, BOARD GAME / ENTITY, PICTIONARY / ENTITY]
Question: How long is the Appalachian Trail?
[HOW / ASSOC, BE / ASSOC, APPALACHIAN TRAIL / ENTITY]
Question: What is the capital of Uganda?
[BE / ASSOC, CAPITAL / ENTITY, OF / ASSOC, UGANDA / ENTITY]
Question: When was Kevin Rudd's wife born?
[WHEN / ASSOC, BE / ASSOC, KEVIN RUDD / ENTITY, WIFE / ENTITY, BEAR / ASSOC]

Query Space Search

The formulation of the simple logic query language along with θ-subsumption allows us to perform searches of the query space in a top down, systematic way via specialisation. The root node is some number of predicates and modifiers with all variables free, and the search subspace is all logic queries θ-subsumed by this base query. Currently the system performs two searches, starting with the following two base logic queries:

$$T(t, s, p, o) \wedge M(m, h, b)$$

$$T(t_0, s_0, p_0, o_0) \land T(t_1, s_1, p_1, o_1) \land M(m, h, b)$$

The search space is finite if the following restrictions are placed upon the search:

- Chunks are used in at most one assignment.
- Unnecessary assignments, such as duplications of existing assignments, are not used.

The state of each node in the search is formulated as a substitution set θ. At the root node this is $\theta = \emptyset$. The successors of a node are found by augmenting the node's θ with possible assignments in turn. The search tree is limited in depth to the number of variables, as each variable can only participate in the left hand side of a single substitution. See Figure 4 for an example of substitution sets at nodes in a search tree. The assignments are also restricted so that a chunk is only assigned to a variable of the same class (with the two classes entity and assoc as above). This is effectively a unary type relation, although it won't be made explicit in the examples that follow. It is also unnecessary to use any chunk in the query more then once, as the same effect can be accomplished by using a variable to variable assignment.

Consider the following question:

Question: What is the capital of Uganda?
[BE / ASSOC, CAPITAL / ENTITY, OF / ASSOC, UGANDA / ENTITY]

The path through the search tree that finds the correct answer is illustrated in Figure 4.

The successors of a node are all specialisations of that node. The space searched using this method forms a graph, which is called the *refinement graph* in ILP parlance. It is a graph because there can be multiple paths to a node, generally those corresponding to reorderings of θ. Because of this, it is possible to reduce the nodes searched by using *memorisation* on visited nodes (a form of dynamic programming), by maintaining a visited node set which new nodes are checked against.

$$T(t,s,p,o) \wedge M(m,h,b)$$
$$\theta = \emptyset$$

$$\downarrow$$

$$T(t,s,p,\text{"capital"}) \wedge M(m,h,b)$$
$$\theta = \{o = \text{"capital"}\}$$

$$\downarrow$$

$$T(m,s,p,\text{"capital"}) \wedge M(m,h,b)$$
$$\theta = \{o = \text{"capital"}, t = m\}$$

$$\downarrow$$

$$T(m,s,p,\text{"capital"}) \wedge M(m,h,\text{"uganda"})$$
$$\theta = \{o = \text{"capital"}, t = m, b = \text{"uganda"}\}$$

$$\downarrow$$

$$T(m,s,\text{"be"},\text{"capital"}) \wedge M(m,h,\text{"uganda"})$$
$$\theta = \{o = \text{"capital"}, t = m, b = \text{"uganda"}, p = \text{"be"}\}$$

$$\downarrow$$

$$T(m,s,\text{"be"},\text{"capital"}) \wedge M(m,\text{"of"},\text{"uganda"})$$
$$\theta = \{o = \text{"capital"}, t = m, b = \text{"uganda"}, p = \text{"be"}, h = \text{"of"}\}$$

Figure 4: An example search path

Ranking Simple Logic Queries

The above method will examine every logic query in the restricted query space previously described. However, it gives no indication as to which query or queries are best able to answer the user's natural language question. In ILP logic programs are chosen based off of their ability to correctly classify positive and negative training examples. A similar approach is not possible with natural language queries, as the answer (the positive example required) is not known. Instead a heuristic approach is needed. The system described here uses information about the structure of the query as well as the number of results returned when it is executed. Evaluating the number of results returned at each node allows us to prune the search space as well, as it is unnecessary to check the children of a node that returns no results (their result set will also be $\emptyset$ as they must be a subset).

Interior nodes (those with children which return more than zero results) of the tree are not considered for the set of candidate queries either. The intuition is that if the answer to the natural language question is in an interior node then it is likely also in the children of that node. If interior nodes were considered the set of candidate queries would be too large, so this distinction is also made by necessity.

Algorithm 2 Ranking of simple logic queries.

Firstly, queries with identical result sets are merged together, with a *occurrence* count kept to record how many were merged into each query.
Queries are then sorted in ascending order by the sum of the following, with the highest ranked of each merged set chosen to represent that set:

- $-\frac{1}{2}$ times the number of results.
- 1 times the search depth.
- -4 times the number of predicates (this puts the 1 triple 1 modifier queries on equal footing with the 2 triple 1 modifier ones).
- 1 times the number of occurrences.

The ranking algorithm is just a tuned heuristic. The idea is that *good* queries return few results (hopefully just the original question's answer), and that more heavily constrained queries tend to return less noise (so answers they give are more likely to be good answers).

Algorithm 3 SQL Database definition (hsqldb syntax).

```
CREATE TABLE triple (
tid identity,
subject varchar(255),
predicate varchar(255),
object varchar(255))

CREATE TABLE modifier (
mid identity,
tid integer,
head varchar(255),
phrase varchar(255))
```

Efficient Simple Logic Query evaluation

Simple logic queries created from substitution sets can be translated into SQL in a straightforward way, See algorithm 3.

Algorithm 4 Simple logical query with substitution set to SQL translation procedure.

Build up a SQL query as *SELECT {selection} FROM {tables} WHERE {clauses}* using the following steps:

1. **Create unique predicate names.** Name the predicates that appear in the query in the order they appear, with a distinction between triples and modifiers (e.g t0, t1,m0,m1,m2).

2. **Create unique variable names.** Name the variables by the location in the predicate they appear in, prefixed by the predicate name in the usual SQL way(e.g t0.object). If a free variable appears in multiple predicates, only include the first.

3. **selection** is a comma separated list of the names of each free variable.

4. **tables** is a comma separated list of the tables and predicate names in usual SQL notation (e.g triple as t0, triple as t1, modifier as m0 ...)

5. **clauses** is a *AND-ed* comma separated list, one item per set of assignments. An assignment is mapped to SQL as a variable assignment. A set of assignments is just a parenthesized expression of the individual assignments, separated by *OR*s.

For example:

$$T(x, y, z)$$
$$\theta = \{x = \text{"john"}, y = \text{"visit"}, z = \{\text{"east germany"}, \text{"germany"}\}\}$$

Translates to:

```
SELECT t0.subject, t0.predicate, t0.object
FROM triple as t0
WHERE
(t0.subject='john') AND
(t0.predicate='visit') AND
(t0.object='east germany' OR t0.object='germany')
```

A Side Note: Triple Queries & Inference

Consider a variation of the question discussed in the introduction:

When was Kevin Rudd's wife born?

The answer was contained in the following snippet of text from a corpus:

> Therese Rein (born 17 July 1958) is an Australian businesswoman and the wife of the 26th Prime Minister of Australia, Kevin Rudd.

The two relevant facts in this sentence are

Therese Rein is the wife of Kevin Rudd.

Therese Rein was born on 17 July 1958.

To answer the question, these two facts need to be combined. Inference of this kind could be preformed by deductive databases or logic programming systems if the facts were stated in an appropriate way. The system just described does not perform this sort of inference, although the effect is duplicated. If only two facts (represented by individual triples) need to be combined, then the two triple, one modifier query space will contain a query that combines these two facts. Since a query involving a join is typically more constrained then other queries, the result ranking algorithm will also place it high in the results.

Results

Corpus & Questions

The test corpus consisted of 20 documents, each on a different subject. The documents were selected so that each of the 20 questions had a document containing information relevant to answering that question. The total corpus size was 7,916 words, totaling 44.3Kb, which results in 1612 triples and a 146Kb triplestore when it is serialised to comma separated value format. The documents were chosen to contain paragraphs representative of what a information retrieval system would select based off of a keyword search. This selection was done by hand. If an information retrieval system was used as a front end the conditions would be more favorable as only triples over a single topic (rather then 20 topics) would need be used at a time. Paragraphs were excepted from web pages appearing on the first page of a Google search for relevant keywords from the questions, as well as the Wikipedia pages on the topics. The triple extraction process took approximately 309 seconds, mostly due to the speed of the Stanford parser. All timed runs were obtained on a 2.5GHz Core2 Duo machine, with 2GB of RAM.

The questions were selected from the TREC8 (Voorhees and Harman, 2000) training question set. The first 20 questions were selected, with the exception of several questions for which corpus documents could not be compiled. As a different corpus than the one the questions were compiled for was being used, this was necessary. The running times for answering each question are shown in Figure 5.

The system was also compared against a simple wh-word substitution question answering system over the same triple database, which formed the baseline. *wh-word* substitution is a primitive form of direct translation to queries and it was not expected to do well.

Answer Evaluation

The system was configured to output the combined results for each of the leaf queries it found, displayed in the order the queries were ranked. Each question was checked against the results to determine 3 things:

1. Did the answer occur in the results anywhere?
2. Did the answer occur in the top 5 results?
3. Did the answer occur in the top 20 results?

Table 2 gives these results. Of the 20 questions, 10 were answered, and 7 had the answer in the top 5 results. In all 7 of these cases, it was found that the correct answer was clear from the results (i.e results ranking above the correct one were of the wrong type). The average number of results displayed was 32.6, with standard deviation 23.7, being displayed as a result of an average of 19.2 leaf queries per question. The average running time for 1 triple 1 modifier queries was 3.38 seconds, whereas the average total running time was 15.2 seconds.

The baseline system was able to answer 3 of the 20 questions. It managed to answer the two subjectively easiest questions (numbers 19&20), as well as question 13.

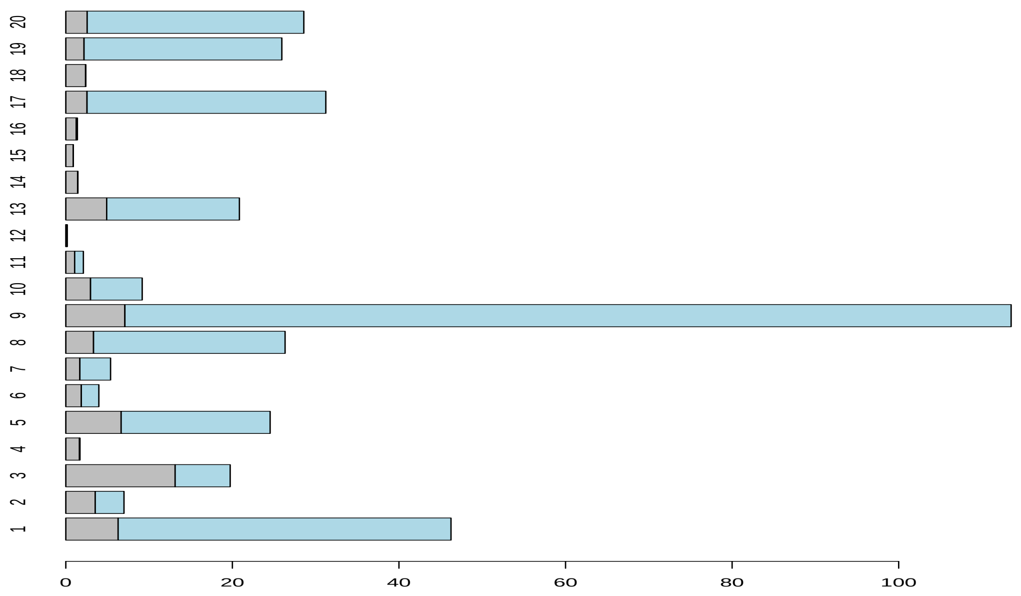

Figure 5: Running times in seconds for the 20 test questions. 1 modifier 1 triple queries shown as subset of time in gray.

n	chunks	stime	time	depth	searched	leafs	results	had answer	top5	top20
1	7	6281	46250	5	6990	75	69			
2	5	3531	6985	5	396	10	8			
3	12	13125	19735	5	3336	21	28			
4	5	1641	1704	5	142	3	13	✓	✓	✓
5	13	6641	24532	6	5084	53	39	✓	✓	✓
6	10	1860	3969	6	1235	12	18			
7	4	1688	5375	4	525	17	44			
8	5	3328	26328	6	1833	23	23			
9	8	7094	113500	6	6669	24	51	✓		✓
10	8	2968	9172	5	1115	25	33	✓		✓
11	3	1062	2109	4	494	19	59			
12	3	140	156	1	14	0	0			
13	8	4907	20844	6	2595	34	89	✓	✓	✓
14	3	1422	1454	4	92	4	6			
15	3	891	891	4	66	4	20	✓	✓	✓
16	3	1250	1390	4	86	4	12			
17	4	2547	31219	6	852	17	44	✓		✓
18	5	2375	2391	5	111	2	4	✓	✓	✓
19	4	2188	25938	6	660	17	48	✓	✓	✓
20	4	2562	28578	6	1095	19	43	✓	✓	✓

Table 2: Information on the runs for the 20 questions. *stime* is running time for the 1 triple 1 modifier query search, *time* is the total search time.

n	Question
1	What date in 1989 did East Germany open the Berlin Wall?
2	What is the shape of a porpoises' tooth?
3	What is the number of bison thought to have been living in North America when Columbus landed in 1492?
4	The Faroes are a part of what northern European country?
5	The symptoms of Parkinson's disease are linked to the demise of cells in what area of the brain?
6	What hotel was used for a setting of the Agatha Christie novel, "And Then There Were None"?
7	What year was the Magna Carta signed?
8	Who was Lincoln's Secretary of State?
9	How long does it take to fly from Paris to New York in a Concorde?
10	How many lives were lost in the Pan Am crash in Lockerbie, Scotland?
11	What is the world's population?
12	Which atlantic hurricane had the highest recorded wind speeds?
13	How long does it take to travel from London to Paris via the Chunnel?
14	Who created the board game Pictionary?
15	How long is the Appalachian Trail?
16	What was Malcolm X's original surname?
17	What is the capital of the United States?
18	How tall is the Statue of Liberty?
19	What is the capital of Uganda?
20	What is the capital of Wisconsin?

Table 3: Mixed topic corpus questions.

Discussion & Future Work

One interesting outcome of the testing is the success of the result ranking algorithm. Given that it involves no semantic analysis of the question, the fact that 7 out of 10 times it places the correct answer in the top 5 is quite remarkable. Unfortunately, in most of the cases the correct answer was not first. A comparison with the answers placed before the correct answer suggests that the correct answer would be clear to a user if they were given a list of the top 5 answers. This suggests that questions and answers need to be analysed for type information. For example, a question that is asking for a place name would only be shown answers that a named entity extractor found were place names. Practical question type analysis systems have shown accuracy above 98% (Li and Roth, 2002), so the success of such a method would depend almost entirely on the accuracy of the named entity extractor used.

Machine Learning approaches to result ranking are widely used in other question answering systems and would likely be a good fit here as well. Result ranking of triple queries is a significantly different problem from directly ranking answers, and is an interesting avenue for future research. The success of the heuristic method used here, which relies only on triple query structure, suggests that the extra information in the triple queries would allow for better ranking.

Although no empirical evaluation of the triple extractor was performed directly, some notable deficiencies became apparent over the course of development of the system. The main disadvantage of a pipeline system is the compounding of errors. Each stage of the system has an error rate associated with it, and when the errors of each stage are multiplied, it is found in practice that few sentences are analysied correctly by all stages. One of the advantages of the triple knowledge representation is a robustness to these errors. However, errors in named entity extraction and coreference resolution were found to contribute to many incorrect triples. The use of a full coreference system was probably unnecessary, most of the cases where coreference was useful an anaphora resolution system would have provided the same benefit. Parse errors were also a common problem, although this was mostly alleviated by the use of the top 3 parse trees for each sentence, rather then just the most likely. The increase in recall this provided was found to outweigh the noise it introduced in the results.

The natural next step in evaluating the system discussed would involve integrating it with an information retrieval system, so that larger corpora could be used. This would allow a direct comparison against other research question answering systems using the TREC guidelines (Dang et al., 2007). Rather then preforming logical queries over the entire triple database, which would be prohibitively large for the TREC corpora, queries could only be preformed over triples associated with the top 20-50 documents returned by the information retrieval system. If this was done on a per question basis it would keep the search tractable.

The largest contributing factor to the speed of the current algorithm was the need to evaluate each query during the query space search to determine if any results were returned. The full query was run at each node, but the results were only used to determine if the query had zero results or not. A system using a customized query for this purpose would be much faster, likely by a factor of 10 or more. A caching strategy for intermediate queries could reduce this even further. Rather than search the entire subspace, a heuristic search could be used such as a beam search. It is not clear what heuristics would be effective, or indeed if a heuristic would work at all.

The baseline system achieved 15% accuracy. Since it is a method of direct translation from questions to logical queries, it was expected to return the correct answer or no answer at all. This was the case, in fact no results were returned at all for questions that it did not answer correctly. As far as question answering systems go, it was very primitive. Unfortunately, the author was unable to find any other question answering systems over triple knowledge bases for which binaries or source code was available.

Related Work

The triple knowledge representation has its root in linguistics, where the subject-predicate-object terminology comes from. The first account of the importance of these three components and how they are a universal attribute of languages was in Greenberg (1963). Triples were used as a knowledge representation language in some early expert systems, particularly in medical applications (Warner, Olmsted, and Rutherford, 1972). Although more expressive knowledge formulations have been more successful such as description logics (Baader, Calvanese, McGuinness, Nardi, and Patel-Schneider, 2003). Triples have since made a resurgence in user annotated meta-data, particularly in the form of the Resource Description Framework (rdf, 2004), which forms the core technology behind the Semantic Web . The approach of treating the subjects and occasionally the objects as resources or entities leads naturally the formulation of semantic graphs (Sowa, 1984), where the subjects and objects of triples are treated as nodes and edges are labeled with the predicates. Question answering systems built on semantic graphs have been explored (Dali, Rusu, Fortuna, Mladenic, and Grobelnik, 2009), although only using very limited natural language querying support (limited to questions that match predefined language templates).

A variety of different approaches have been used to represent modifiers or additional information in triples. Katz (1997) used a nested triple system, which allows for much more complex knowledge to be represented then plain triples. The method in Rusu et al. (2007) only allows for nesting of modifiers with other modifiers, which is still significantly more general then the approach described in this report.

In connection with the resurgence in interest of triples as metadata, a variety of approaches for machine generating triples from natural language have been explored. In Rusu et al. (2007) heuristics for triple extraction are specified over a variety of different types of parse trees. Nakayama (2008) uses similar heuristics, although both methods are significantly simpler then the approach described here. Dali and Fortuna (2008) uses a Support Vector Machine for triple extraction, using a binary classification approach to pick a subset from the large number of possible triples that could be extracted form the sentence.

Many real world systems have taken more general logic formulations as their knowledge representation language. Techniques include heuristic (Elworthy, 2000) and machine learning (Zettlemoyer and Collins, 2009) approaches. Molla, Aliod, Berri, and Hess (1998) used Horn clauses as the knowledge representation language and used a bottom up query expansion method.

As far as the author is aware, inductive logic programming techniques have not previously been applied to direct question answering. Although the approach discussed in this report uses similar techniques and notation, it does so to very different ends. ILP is a form of learning as opposed to an inference system. The technique of *top-down search of refinement graphs* was pioneered by the ILP system MIS (Shapiro, 1982).

Conclusions

The natural language question answering system described showed promising results on the test corpus, with 35% of the questions having their correct answer in the top 5 answers returned.

It was not initially clear from the formulation of the logic query space that an exhaustive search would be practical, but the results confirm that it is able to finish in a reasonable amount of time. The single triple and modifier query search took on average 3.38 seconds, which with suitable optimisation is fast enough for use in practical systems. The two triple, one modifier query search often took substantially longer, with an average of 15.2 seconds, although the median was only 4.95, suggesting most of that time was taken on a few long running queries. The average size of the query spaces was 1670 queries, a reasonable size for exhaustive search methods to deal with.

The triple extraction method worked well in practice, although it was prone to oversimplifying sentences, and was found to be vulnerable to errors in the language pipeline, notably in the named entity recognition and coreference resolution stages.

Given the search method's radical departure from existing methods, and the short time span it was built in, its performance suggests it would perform well as the core of a more refined question answering system, or as part of an ensemble of methods.

Bibliography

Resource description framework, 2004. URL http://www.w3.org/RDF/.

Semantic web. URL http://www.w3.org/2001/sw/.

Wikipedia:redirect. URL http://en.wikipedia.org/wiki/Wikipedia:Redirect.

Steven P. Abney. Parsing by chunks. In *Principle-Based Parsing*, pages 257–278. Kluwer Academic Publishers, 1991.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*, 2003. Cambridge University Press. ISBN 0-521-78176-0.

Ann Bies, Mark Ferguson, Karen Katz, Robert Macintyre, Victoria Tredinnick, Grace Kim, Mary Ann Marcinkiewicz, and Britta Schasberger. Bracketing guidelines for treebank ii style penn treebank project, 1995.

Lorand Dali and Blaz Fortuna. Triplet extraction from sentences using svm. In *SiKDD*, 2008.

Lorand Dali, Delia Rusu, Blaz Fortuna, Dunja Mladenic, and Marko Grobelnik. Question answering based on semantic graphs. In *SemSearch*, 2009.

Hao Trang Dang, Diane Kelly, and Jimmy Lin. Overview of the trec 2007 question answering track. In *The Sixteenth Text REtrieval Conference (TREC 2007) Proceedings*, 2007.

David Elworthy. Question answering using a large nlp system. In *TREC9*, 2000.

C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

Joseph H. Greenberg. *Universals of language*. MIT Press, 1963.

Vera Hollink, Jaap Kamps, Christof Monz, and Maarten de Rijke. Monolingual document retrieval for european languages. 2003.

Boris Katz. Annotation the world wide web using natural language. In *RIAO*, 1997.

Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 (NIPS*, pages 3–10. MIT Press, 2003.

Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA, 2002. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1072228.1072378.

Julie Beth Lovins. Development of a stemming algorithm. *Mechanical translation and computational linguistics*, 11:22–31, 1968.

Christopher D. Manning and Hinrich Schiitze. *Foundations of statistical natural language processing*. MIT Press, 1999.

Christopher D. Manning, Prabhakar Raghaven, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Diego Molla, Diego Molla Aliod, Jawad Berri, and Michael Hess. A real world implementation of answer extraction. In *In Proceedings of the 9th International Workshop on Database and Expert Systems, Workshop: Natural Language and Information Systems (NLIS-98)*, pages 143–148, 1998.

Kotaro Nakayama. Wikipedia mining for triple extraction enhanced by co-reference resolution. In *SDoW*, 2008.

Delia Rusu, Lorand Dali, Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. Triple extraction from sentences. In *SikDD*, 2007.

Ehud Y. Shapiro. *Algorithmic program debugging*. PhD thesis, New Haven, CT, USA, 1982.

John F. Sowa. *Conceptual structures: information processing in mind and machine*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984. ISBN 0-201-14472-7.

Zareen Syed, Tim Finin, and Anupam Joshi. Wikitology: Wikipedia as an ontology. In *Proceedings of the Grace Hopper Celebration of Women in Computing Conference*. ACM Press, October 2008. (poster paper).

Yannick Versley, Simone Paolo Ponzetto, Massimo Poesio, Vladimir Eidelman, Alan Jern, Jason Smith, Xiaofeng Yang, and Alessandro Moschitti. Bart: A modular toolkit for coreference resolution. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*, 2008.

Ellen M. Voorhees and Donna Harman. Overview of the eighth text retrieval conference. In *TREC-8*, 2000.

H Warner, C Olmsted, and B Rutherford. Help–a program for medical decision-making. *Computers and biomedical research*, 1972.

Luke S. Zettlemoyer and Michael Collins. Learning context-dependent mappings from sentences to logical form. In *ACL-IJCNLP*, 2009.

Appendix A : Algorithms

Algorithm 5 Conjunction splitting pass pseudo-code.

```
def SplitCoordinatingConjunctions( t : TreeWithModifiers ) = {
    if one of t's children is a CC node {
        var replacements = splitOnCCs(t)
        return replacements flatMap (SplitCoordinatingConjunctions)
    }

    var results = []
    for( child <- t.children ) {
        var replacements = SplitCoordinatingConjunctions(child)
        var clones = []
        for(new_child <- replacements)
            clones += A clone of t with child replaced with new_child

        if( clones.size > 1 )
            results += clones flatMap (SplitCoordinatingConjunctions)
    }

    return results
}

def splitOnCCs( t : TreeWithModifiers ) = {
    var new_children = []
    var forests = splitChildrenOfTree(t, {CC, ","})
    for ( forest <- forests ) {
        group = new node with same label as t
        group addChildren forest
        new_children += group
    }

    return new_children
}

def splitChildrenOfTree( t : TreeWithModifer, delims : String) = {
    val forest = []
    foreach consecutive group g between delims {
        val n = new Node with class t.class
        n.children += g
        forest += n
    }

    return forest
}
```

Algorithm 6 Dependent clauses extracting pass pseudo-code.

```
val adj_heads = {WHNP, WHADJP, NP, WP$, WP, WDT}
val advp_heads = {WHADVP, WHAVP, IN, WHPP, WRB}

def ExtractDependentClauses( t : TreeWithModifiers) = {
    var results = []

    if t in {S, FRAG, SBAR, PRN}
        excise t from its parent

    if t in {S, FRAG}
        results += t

    if s is SBAR {
        var subclause = first child of t matching S
        if subclause != null {
            var adjHead = get child matching adj_heads
            if adjHead != null {
                var subject = first NP in a DFS of parent
                add subject as first child of subclause
                t = subclause
            }

            var advHead = get child matching advp_heads
            if advHead != null
                t += new modifier(advHead, subclause.toString)
        }
        results += t
    }

    if t is PRN {
        var subject = first NP in a DFS of parent
        remove Left and right bracket symbol nodes from t
        results += new tree of form
            (S
                {subject}
                (VP (VBZ be)
                    {t}))

    }

    t.children = t.children flatMap (ExtractDependentClauses)
    return [] if results contains a single tree with no children
    otherwise return results
}
```

Algorithm 7 Adjective phrase extraction pass pseudo-code. See 5 for definition of splitChildrenOfTree.

```
def ExtractAdjectivePhrases( t : TreeWithModifiers) = {
    val forest = splitChildrenOfTree(t, {',', ';', '.'})

    if forest.size == 2 {
        val head = first child of the first tree in forest
        val pred_parent = second tree in forest
        val pred = first child of pred_parent

        if head is NP {
            excise pred from t
            if pred is not NP
                add copy of head as first child of pred_parent

            make pred_parent S
            results += pred_parent
        }
    }

    results += t.children flatMap (ExtractAdjectivePhrases)
    return results
}

def (xs map f) = transform each element of list by applying f
def (xs flatMap f) = map then concat list of lists to single list
```

Algorithm 8 Prepositional Phrase extraction pass pseudo-code.

```
def ExtractPrepositionalPhrases( t : TreeWithModifiers) = {
    val results = [t]
    val pps = removePrepositionalPhrases(t,null)

    if pps != null {
        reverse(pps)
        for( pp <- pps ) {
            val head = head of pp phrase as string
            val body = pp with head excised as string

            if body != null
                t += new Modifier(head,body)
        }
    }

    return results
}

def removePrepositionalPhrases( t : TreeWithModifiers,
                                parent : TreeWithModifiers) = {
    val pps = []

    for( c <- t.children )
        pps += removePrepositionalPhrases(c,t)

    if t is PP {
        if parent != null
            parent.remove(t)
        pss += t
    }

    return pps
}
```