Exploring the Performance of Boolean Retrieval Strategies for Open Domain Question Answering

Horacio Saggion
saggion@dcs.shef.ac.uk

Robert Gaizauskas
r.gaizauskas@dcs.shef,ac,uk

Mark Hepple
m.hepple@dcs.shef.ac.uk

Ian Roberts
i.roberts@dcs.shef.ac.uk

Mark A. Greenwood
m.greenwood@dcs.shef.ac.uk

Department of Computer Science
University of Sheffield
Regent Court, Portobello Road
Sheffield S1 4DP UK

ABSTRACT

This paper is concerned with systematically exploring and evaluating a range of possible boolean retrieval strategies for use within a Question Answering (QA) system. We firstly set out two evaluation metrics — *coverage* and *recall* — which are specifically designed for use in evaluating retrieval performance in a QA context, and apply these measure in quantifying the performance of some standard ranked retrieval systems for this purpose. We then consider a series of possible boolean retrieval strategies for use in QA, which concern the way that boolean queries are generated from questions to retrieve passages relevant to finding the question's answer, and evaluate their performance. This line of research should ultimately lead to an increased understanding of how best to formulate retrieval strategies for QA and of which component methods can usefully contribute to such strategies.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*

General Terms

Experimentation, Performance, Measurement

Keywords

Question answering, Information retrieval

1. INTRODUCTION

Open domain question answering (QA), as pursued within the TREC QA track [9], is the task of finding exact answers to natural language questions in large text collections. A common approach to building such systems is to couple an information retrieval (IR) engine with an answer extraction (AE) system. The IR system is given a question, or a preprocessed version of it, as a query and returns a relatively small set of candidate answer bearing documents from the large text collection. This document subset is then passed on to the AE component which processes each text in detail and attempts to identify the precise answer to the question. This strategy has the advantage of building on the strengths of IR systems for efficient indexing and search of large collections, while allowing more processing intensive NLP techniques to be applied to the candidate documents believed most likely to contain answers. However, it has the consequence of bounding the overall performance of the QA system by the performance of the IR system: if the IR system returns documents none of which contain the question's answer, then no amount of sophisticated processing on the part of the AE component can remedy this.

It follows that failure analysis of any QA system built according to the two stage IR-AE strategy must pay due attention to analyzing the performance of the IR system *for the task of question answering*. One cannot blindly assume that the best performing IR system, according to some common IR benchmark, e.g. the TREC ad hoc track, will be the best system to use in the QA context.

To investigate the issue of how to optimise the IR component of a QA system we have been engaged in a programme of on-going experimentation with various IR engines and the development of appropriate measures for evaluating IR systems in the QA context. In [6] we introduced two new measures for evaluating IR systems in the QA context and reported the performance, according to these measures, of various passage retrieval strategies within a probabilistic IR model. These experiments showed that for the TREC question set considered the best performing system configuration was one for which at least one answer bearing document was returned within the top 200 ranks for approximately 85% of questions

While these results are helpful, several further problems emerged as a result of this work. First, it underlined the importance of getting more answer bearing documents returned at lower ranks. Although, an answer bearing document was found within the top 200 ranks for 85% of questions, only 65% had answers within the top 20 ranks. Processing more documents per question is costly and like to lead to less accurate question answering, since there are

more distractors for the AE component. For our own AE component, we found that overall QA scores went down when we processed 200 top ranking documents per question as opposed to 20, even though more of the questions had answers within the top 200 (in fact rank 20 turned out to be the optimal setting for our combined IR-AE system [2]). Secondly, working with a probabilistic model is frustrating during failure analysis, as the behaviour of such models is relatively opaque. For the quesion *When did the shootings at Columbine happen?*, for example, the state-of-the-art probabilistic engine we used did not return an answer bearing passage within the top 200 passages, despite the fact that there is a document which contains the string *Families of 10 victims of the April 20, 1999, shootings at Columbine High School ....* Such behaviour is the result of a complex interaction between a probablilistic term weighting measure and characteristics of the specific collection being searched, something which is virtually impossible to disentangle in any given case, and unlikely to lead to any generalisable insight.

To address these problems, and because of positive results from using boolean IR for QA reported by [3, 4], we decided to carry out experiments using boolean retrieval with various strategies for deriving boolean queries from questions and for assembling candidate answer bearing passage sets. In particular we wanted to know (a) How does a boolean IR system given a conjunction of question words as a query perform in terms of returning answer bearing documents? (b) If a conjunction of question words is too restrictive as a query (i.e. returns too few documents) how best should the query be weakened? (c) If the correct documents are not being returned regardless of which question words are dropped from the query, how should the query best be expanded to retrieve more answer bearing documents? – e.g. can disjoining morphological variants or synonyms of question words help? (d) What is the effect of varying the match window size – the number of sentences within which the boolean query must match? (e) What is the value of looking for answers in a passage broader that the match window? These questions have been investigated to some extent by others, but to our knowledge no one has explored them in the QA context (i) using measures specifically designed to assess the utility of an IR system for QA (ii) assembled as much empirical evidence as we have here.

The paper is structured as follows. In the next section we introduce the evaluation measures we use throughout the paper and describe the data set against which all our experiments have been run. Section 3 presents results from using several ranked retrieval engines "off-the-shelf" – these provide baseline figures in the sense of figures which need to be beaten by any effort to customize IR for QA. In section 4 we describe previous work that has used a boolean search strategy in a QA context. Section 5 describes the actual experiments we have run and their results. We conclude with an analysis and discussion of these results.

2. EVALUATING RETRIEVAL FOR QA

2.1 Evaluation Measures

As noted above, one cannot assume that the best performing IR system as measured by a standard benchmark such as the TREC ad hoc task will be the best performing system for the open domain QA task. In [6] we argue for the use of two measures which we believe are more helpful in capturing aspects of IR system performance of relevance in the QA setting than the conventional metrics of recall and precision. These metrics are *coverage* – the proportion of the question set for which a correct answer can be found within the top n passages retrieved for each question and *answer redundancy* – the average number, per question, of passages within the top n ranks retrieved which contain a correct answer (alternatively: how many chances per question on average an answer extraction component has to find an answer). We do not reproduce the arguments for the merits of these measures here – the reader should refer to the aforementioned paper for further details. Formally these measures are defined as follows. Let Q be the question set, D the document (or passage) collection, $A_{D,q}$ the subset of D which contains correct answers for $q \in Q$, and $R^S_{D,q,n}$ be the n top-ranked documents (or passages) in D retrieved by a retrieval system S given question q. The *coverage* of a retrieval system S for a question set Q and document collection D at rank n is defined as in (1). The *answer redundancy* of a retrieval system S for a question set Q and document collection D at rank n is defined as in (2). The *actual redundancy* $\sum_{q \in Q} |A_{D,q}|/|Q|$ is the upper bound on the answer redundancy achievable by any QA system. Comparing answer redundancy with actual redundancy captures the same information that recall traditionally supplies.

$$coverage^S(Q, D, n) \equiv \frac{|\{q \in Q | R^S_{D,q,n} \cap A_{D,q} \neq \emptyset\}|}{|Q|} \quad (1)$$

$$redundancy^S(Q, D, n) \equiv \frac{\sum_{q \in Q} |R^S_{D,q,n} \cap A_{D,q}|}{|Q|} \quad (2)$$

Note that coverage and redundancy are defined in terms of rank, so it is not immediately obvious how they can be applied to boolean retrieval approaches, whose output is inherently unranked. There are several ways to address this issue. One might compute an *unranked* version of the metrics, e.g. with coverage being the proportion of questions for which a correct answer is found among the results returned under the given boolean strategy (and analogously, redundancy). This is simple, but has the disadvantage that there is no easy way to compare coverage and redundancy figures between boolean and ranked approaches at a given rank (though one possibility would be to compute the average number of passages returned per question by a boolean approach and compare performance with the ranked approach at the rank equal to this average value).

Alternatively, one can order the results of a boolean-based strategy post-hoc by applying some ranking function to the returned passages, allowing the coverage and redundancy measures to be employed as defined above. Since the number of results returned for boolean retrieval is highly variable, the results for a given question may number less than the rank r for which coverage is being assessed. Consequently, coverage and redundancy will be computed over *at most* r results per question for a boolean approach, and the resulting scores should be interpreted in this light when compared to those for a ranked retrieval approach for which *exactly* r results will be provided per question.

In the experiments reported later in the paper, we adopt a post-hoc ranking approach to enable us to generate coverage and redundancy figures for boolean strategies. In one sense this turns a boolean approach into a ranked retrieval one. However, by separating question manipulation, searching and result ranking we obtain a more fine-grained instrument for examining how retrieval for QA is working or failing than by rolling all three into one, as happens if a question is fed directly to a more conventional ranked retrieval system of the sort described in Section 3 below.

2.2 Evaluation Dataset

The data used in the experiments reported in the rest of this paper is that used in the TREC2003 QA track – see [9] for more details. The text collection is the AQUAINT corpus consisting of approximately one million documents drawn from three newswire sources for the period 1998-2000 (about 3.2 gigabytes of text). The TREC 2003 question set consists of 500 questions of which 413 were factoid questions and the rest were list questions or definition questions. Here we work only with the factoid questions. For 51 of these factoid questions no answers were judged by human assessors to be both correct and supported by a document in the collection. In the following we exclude these questions in the calculations of coverage and redundancy, meaning that our question set consists of 362 questions. Excluding nil answer questions has the positive effect that coverage scores range from 0 to 100 and are comparable across question sets. How best to assess IR systems for their contribution to identifying questions with no answer remains to be addressed.

In order to support automated evaluation, NIST produces regular expression patterns for each question which match strings containing the answer. Any string drawn from the test collection which matches an answer pattern for a question is said to be correct according to a *lenient* criterion for correctness. If in addition a string matching an answer pattern is drawn from a text which has been judged by a human assessor to support the answer, then the answer is said to be correct according to a *strict* criterion for correctness.

We have estimated the actual redundancy of this text collection plus question set to be 3.68, based on taking the average number of texts per question judged by human assessors to support an answer. This figure is an estimate as a) some supporting documents may contain more than one occurrence of the answer, and b) not every document supporting an answer is likely to have been found by the assessors.

3. RANKED RETRIEVAL BASELINES

Ranked retrieval approaches to IR have become the preferred approaches within the IR research community, having conclusively established their superiority on a variety benchmark tasks. Furthermore a number of these systems are available to researchers "off-the-shelf". It is natural, therefore, to assess these systems for the QA task given the simple but widely used approach of using the question unmodified as the IR query. These results provide a baseline against which the results of experiments with boolean approaches can be compared.

3.1 Okapi

Okapi and especially its `bm25` weighting scheme have often been regarded as representing the state-of-the-art in ranked retrieval [7]. Our experiments using Okapi relied on its native passage retrieval support which according to [8] works as follows. All passaging is done at search-time, not at index time. Passages are based on paragraph boundaries, and the experiments in this paper all use passages which are one paragraph in length. Okapi's notion of paragraph is not entirely clear – it does not correspond to the embedded SGML paragraph tags in the corpus – but empirical examination suggests average paragraph length of about four sentences. Given a query Okapi first treats each document as a single passage and considers all documents whose weight exceeds a threshold set empirically as the weight of the 10,000th document. The documents above this threshold are then broken into passages and each passage is scored. The initially retrieved documents are then re-ranked according to the score of their best passage, and the single best passage from each document is returned. The performance of Okapi on the evaluation dataset described in Section 2.2 above is shown in Table 1.

Table 1: Okapi's performance over the question set used in the TREC2003 QA evaluation.

	Strict		Lenient	
Rank	coverage	redundancy	coverage	redundancy
1	21.5	0.21	30.1	0.30
5	39.5	0.56	50.8	1.04
10	48.1	0.80	60.5	1.75
20	55.0	1.08	69.3	2.93
30	61.6	1.28	74.3	3.89
50	67.7	1.50	80.4	5.64
100	71.8	1.76	84.3	8.89
200	78.2	2.06	89.8	14.32

3.2 Lucene

Lucene, a freely available open-source IR engine, supports a boolean query language while performing ranked retrieval using the standard `tf.idf` weighting scheme with the cosine similarity measure[1]. In our experiments with Lucene the AQUAINT collection is split into passages (using the paragraph markers in the source documents) and the resulting paragraphs have stopwords removed and are stemmed using the Porter stemmer [5] before being indexed. Average paragraph length is about 1.5 sentences.

Passages are retrieved using queries consisting of all the question words (Lucene subsequently removes stopwords and stems remaining terms before accessing the index). The ranking algorithm prefers short passages containing all the question words over longer passages and those which do not contain all the question words. The coverage and redundancy figures obtained using Lucene as described can be seen in Table 2.

3.3 Z-PRISE

The final ranked retrieval engine considered was Z-PRISE, a vector space retrieval system freely available from the National Institute of Standards and Technology (NIST) [1]. For this engine, a set of results have been provided by the TREC

[1] `http://jakarta.apache.org/lucene/`

Table 2: Lucene's performance over the question set used in the TREC2003 QA evaluation.

	Strict		Lenient	
Rank	coverage	redundancy	coverage	redundancy
1	11.9	0.11	17.1	0.17
5	29.3	0.42	38.1	0.77
10	40.1	0.68	51.7	1.42
20	47.2	0.91	56.9	2.32
30	55.0	1.14	65.2	3.16
50	60.2	1.39	72.1	4.52
100	67.7	1.82	79.6	7.31
200	72.9	2.19	84.3	11.79

QA organisers, providing the top 1000 document identifiers retrieved for each question, using the question as the search query.

One significant difference between any results obtained using Z-PRISE compared to Okapi and Lucene is that we are now evaluating whole documents rather than single paragraph passages (documents in AQUAINT average around 24 sentences). This means that at any given rank an answer extraction system using the results from Z-PRISE would be processing a much greater amount of text than one using the passage-based configuration of Okapi or Lucene as described above, which is costly and brings a risk of lower performance due to the presence of more distractors. On the other hand, this use of whole documents rather than single paragraph passages also means that Z-PRISE ought to have an advantage over Okapi and Lucene with respect to coverage and redundancy measures because the items being assessed at each rank are much longer and hence more likely to be answer bearing. The coverage and redundancy figures obtained using Z-PRISE can be seen in Table 3.

Table 3: Z-PRISE's performance over the question set used in the TREC2003 QA evaluation.

	Strict		Lenient	
Rank	coverage	redundancy	coverage	redundancy
1	21.3	0.21	37.6	0.37
5	40.9	0.60	63.5	1.51
10	50.0	0.83	71.8	2.53
20	62.2	1.18	78.5	4.39
30	67.4	1.37	81.5	5.81
50	71.8	1.67	84.5	8.40
100	76.5	1.93	88.4	13.45
200	80.4	2.17	90.9	21.40
500	84.3	2.48	94.8	41.11
1000	87.0	2.73	95.9	68.63

4. BOOLEAN RETRIEVAL FOR QA

Using boolean retrieval in a QA system immediately throws up some problems that do not arise with a ranked retrieval approach. With ranked retrieval, we can simply take the words of the question and use them as a query, and the engine will retrieve a plentiful supply of candidate passages ranked by their similarity to the question. If, in a similar fashion, we take the words of the question, perhaps excluding stop-list items, and conjoin them to form a boolean query, we may well retrieve no passages at all, as it is quite likely that this precise combination of terms does not appear together in any passage of the entire collection. In this case, we might try to 'weaken' the query so that some passages are retrieved, either by deleting some terms or otherwise generalising them, but there are then choices to be made as to what operations should be applied to which terms. This suggests that boolean retrieval can only be used in combination with a *strategy* for forming retrieval queries from questions, which may need to be a *dynamic*, i.e. which sequentially modifies the query until a promising set of passages are returned. Another problem that arises for a boolean approach is that a retrieval may yield more passages than can reasonably be processed in subsequent answer extraction, but since the retrieved passages are not ranked it is not immediately obvious which to choose for further analysis.

Boolean retrieval has previously been used for question answering in the MURAX system of [4] and the Falcon system of [3]. Kupiec's MURAX system, which predates the TREC QA track, answers general knowledge questions using Grolier's on-line encyclopedia as a text corpus in which to seek answers, which are assumed to be noun phrases. The retrieval engine used in MURAX allows queries to be specified as a conjunction of terms, which may be restricted to appear separated by no more than a specified number of other terms, and may also be required to appear in a stipulated order. MURAX first analyses the question to locate noun phrases and main verbs, and forms an initial query that is quite specific and restrictive, e.g. requiring noun phrase words to appear in strict order and adjacently. Depending on the number of passages returned, new queries are created to either reduce the number of hits (called *narrowing*) or increase them (*broadening*). Narrowing may be done by adding extra terms from the question (e.g. adding main verbs to a query based on noun phrase terms), whilst broadening might be done by loosening order/adjacency requirements on terms, or by dropping words within phrases or entire phrases. This process stops when the number of passages returned falls with a specified range or no further queries can be formulated. The returned passages are then ranked by their overlap with the initial question.

The Falcon QA system [3] uses the SMART retrieval engine to retrieve passages for answer extraction. The initial query is formulated using keywords from the question. This query may then be refined, again with the aim of achieving a number of retrieved documents that falls within a specified range. The operations to modify the query include disjoining a term with various alternates, i.e. so that a query subterm *w1* is replaced by *(w1 OR w2 OR w3)*, including (i) morphological alternations (e.g. *invent* and *inventor* for *invented*), (ii) lexical alternations (e.g. *assassin* for *killer*) and (iii) semantic alternations (e.g. *prefer* for *like*).

Inspired by this previous work, we decided to try using a boolean retrieval approach for one of Sheffield's entries to the TREC-2003 QA Track, using an in-house boolean search engine (*MadCow*) and with the window size for both matching and for passages returned is individual sentences. The query formulation approach used in this work is one giving priority to name expressions found in the question. A name processing component proposes a boolean query condition for each recognised name based on the the name and

any variants of it known to the system, e.g. recognising *Bill Clinton* might produce a condition `(( Bill & Clinton ) | ( President & Clinton ))`. If retrieval using name conditions fails to return enough documents, or returns too many, then further conditions based on the non-name words in a question may be formulated (either to extend an overly weak name condition, or to be used in place of any name conditions). Such conditions are built by expanding the non-stoplist words from the question with known variant forms (i.e. inflectional variants, e.g. *decided/decide*, and related nouns/verbs, e.g. *decision/decide*), and conjoining the results together. If this condition is too restrictive, it is weakened by deleting the sub-expression for the question word having the highest document frequency in the corpus (which is thereby deemed to be the 'least informative' term). This overall process is continued until enough documents have been collected, or the process is exhausted, with the results being ordered firstly by the stage in the sequential process at which they were retrieved and secondly by overlap with the question.

Sheffield's TREC 2003 entry using boolean retrieval produced results that were slightly better than another Sheffield entry using Okapi, but this may be due to passage size effects (passages returned by Okapi were typically more than single sentences, and so may have contained more 'distractor' material on average). We were not able at the time to do any significant evaluation of whether boolean retrieval was returning more answers at lower ranks than Okapi or not. Neither did we evaluate whether specific characteristics of the boolean approach (e.g. the role of name identification) produced better results than other possible formulations. The experiments described hereafter in this paper begin the task of systematically exploring such issues.

5. EXPERIMENTS AND RESULTS

The experiments we present here aim at reinforcing our understanding of query formulation techniques for question answering. They explore different issues that arise in the context of boolean search including:

- *question analysis*: is it worth doing linguistic analysis of the question? if so, what forms of analysis are useful?
- *term expansion*: which strategy should be pursued? synonymy expansion or morphological variants?
- *query broadening*: are measures of a term's discriminative power of use when broadening the search query? if so, is it better to prefer terms of high or low discrimination when selecting terms to discard?
- *passage and matching-window size*: is there any benefit in matching boolean queries in windows that are larger in size than a single sentence, and is there any benefit in returning additional text around the window in which matching is constrained to occur?
- *ranking*: how should one rank sentences returned in a boolean environment, so that the best possible sentences are given first to the answer extraction component?

5.1 Minimal Strategy

The simplest approach for formulating retrieval queries from questions is to use a conjunction of the question terms: we call this configuration *AllTerms*. The question is tokenised and all stop words and words not in the collection removed. The remaining terms are conjoined together to form the retrieval query. For example, the question *How far is it from Earth to Mars?* yields the query *(Mars & Earth)*. The retrieval results for such queries are then ranked using a simple measure of idf-weighted overlap with the question, which we might formalise as: $score(P,Q) = \sum_{t \in Q \cap P} idf(t)$, where Q and P represent the set of non-stoplist terms appearing in the question and passage, respectively. We use this same simple ranking approach in the experiments that follow, except where explicitly stated otherwise.

The results for this configuration are presented in Table 6. With this configuration 178 out of 362 questions return non-empty result sets. The low coverage numbers do not come as a surprise since the requirement that the precise words of the question should appear together in an answer bearing passage is clearly too stringent, but they give us a lower bound for the performance of any other strategy we might try. Table 4 shows the mean and median for the number of sentences returned per question. The mean number of sentences returned per question by *rank* is given in Table 5, where each entry is the mean number of sentences an answer extraction system would have to process for rank k. For *AllTerms* run this number is low, e.g. $\sim$21 sentences for $k = 200$. This contrasts clearly with systems that always return k passages, where answer extraction would have to process $avg\_size * k$ sentences per question, where *avg_size* is the average passage size.

5.2 Simple Term Expansion

If no sentence matches a given question it may be the case that replacing query terms by related terms would bring back matching sentences. We explore two options.

The *WordNet* configuration explores using synonymy expansion to overcome the keyword-barrier. The query obtained with configuration *AllTerms* is used first, but if no matching sentence is found, a broadening technique is applied by which each term of the query is replaced by a disjunction of its synonyms. The method is brute-force, in that no word sense disambiguation is attempted. As an example, the question *What Ridley Scott movie is set in 180 a.d.?* found no matching sentences with configuration *AllTerms*, but synonym expansion of the term *movie* gives (movie | film | picture | ...), which allows retrieval of a sentence containing the correct answer. Table 6 gives coverage and redundancy figures for this run. While 202 question out of 362 have at least one matching sentence in the collection, little improvement is observed over the *AllTerms* configuration. Paired t-test experiments show no differences in coverage between *WordNet* and *AllTerms*.

The *MorphVar* configuration explores the use of morphological variants of query terms. We grouped terms appearing within the corpus into morphological equivalence classes on the basis of returning the same stem string when the Porter stemmer is applied to them. If the original unexpanded query returns no matching passages, then each query term is expanded with a disjunction of its morphological variants for a second retrieval step. This expansion, for example, allows a sentence containing the answer to *What color is the top stripe of the U.S. flag?* to be found, when *AllTerms* failed, by including the plural *stripes* as a variant of *stripe*.

Table 4: Mean and median number of sentences returned per question

	AllTerms	*WNT*	*Morph*	*DropBig*	*DropSmall*	*BigIte*	*SmallIte*	*StrIte*	*StrIteMorph*	*StrIteMorph20*
mean	224.41	225.06	224.69	252.63	341.62	948.12	1151.25	338.35	623.17	2179.69
median	0	1	1	3	8	4	17	10	11	216

Table 5: Mean number of sentences per question at each rank up to rank 200

Rank	*AllTerms*	*WNT*	*Morph*	*DropBig*	*DropSmall*	*BigIte*	*SmallIte*	*StrIte*	*StrIteMorph*	*StrIteMorph20*
1	0.43	0.51	0.51	0.66	1.00	0.70	1.00	1.00	1.00	1.00
5	1.74	1.95	1.91	2.62	3.97	2.84	4.07	3.99	4.08	5.00
10	2.89	3.15	3.09	4.32	6.65	4.84	7.05	6.77	6.97	10.00
20	4.74	5.06	4.99	7.03	11.01	8.14	12.11	11.22	11.54	20.00
30	6.31	6.66	6.59	9.27	14.71	10.97	16.45	15.02	15.36	29.46
50	8.92	9.31	9.20	12.90	20.86	15.76	24.03	21.26	21.69	46.26
100	13.80	14.31	14.08	19.79	32.73	25.43	39.44	33.43	33.53	81.04
200	20.69	21.35	20.98	29.48	49.72	40.06	62.00	50.90	50.64	136.82

The coverage and redundancy figures given in Table 6 show a better improvement than with *WordNet*. For this configuration, 203 questions out of 362 have at least one matching sentence in the collection. Paired *t*-tests show significative improvent ($p < 0.01$) in coverage at all ranks when comparing *MorphVar* with *AllTerms*.

5.3 Document Frequency Broadening

An alternative approach to query relaxation is the deletion of question terms from the query. Two basic configurations were tested here, which both begin with the *AllTerms* conjunctive query, and modify this only if no matching passage is returned. The *DropBig* configuration discards from the initial query the question term having highest document frequency, i.e. the least discriminative or informative question term, whilst the *DropSmall* configuration discards the question term that has lowest document frequency and hence is most informative. For example, the question *How fast can a king cobra kill you?* has (fast & king & cobra & kill) as its conjunctive query which matches no sentence in the collection. *DropBig* modifies this query to (king & cobra & kill) which retrieves a sentence containing the answer. *DropBig* finds matching sentences for 273 questions. The initial query (dissolves & gold) for the question *What dissolves gold?* matches no sentence. However, the modified query (gold) produced by *DropSmall* is overly general and retrieves a large number of results, although an answer bearing passage is included amongst them. *DropSmall* find matching sentences for 288 questions.

Discarding a term helps, but many questions still do not find matching sentences. To address this problem, an iterative process can be used that sequentially deletes terms until either a matching sentence is found or no more terms can be deleted. The configurations *BigIte* and *SmallIte* perform such iterative deletion, prefering the highest and lowest document frequency term for deletion, respectively. Both configurations produce sentences for all questions. Coverage and redundancy results in Table 6 show that better performance is obtained when the least informative term is prefered for deletion.

Paired *t*-tests show significative improvent ($p < 0.01$) in coverage at all ranks when comparing both *DropBig* and *DropSmall* with *AllTerms*. However, coverage improvement is only significative at all ranks ($p < 0.01$) when comparing *BigIte* to *DropBig*.

5.4 Structure Analysis

As an initial exploration of whether recognising structure within questions can contribute to effective retrieval for QA, we investigate an approach *StrIte* which distinguishes proper names and quoted expressions from other terms in a question, and treats these three groups differently during query formulation and post-hoc ranking of results. The motivation for distinguishing these elements is that where a question contains a proper name, e.g. *What college did Magic Johnson attend?*, one would like to find passages containing these proper nouns more than anything else. Likewise, if a question contains a quoted expression, e.g. *Who starred in "The Poseidon Adventure"?*, those terms typically represent a particular entity such as the name of a book or film or a famous quotation, and one should search for all of them in candidate passages. We perform a simple analysis of the question that extracts quoted expressions, and uses POS tagging to identify proper nouns (i.e. selecting NNPs). For example, analysis of the question *What is Richie's surname on "Happy Days"?* yields the following three term groups:

Name terms:	Richie
Quote terms:	Happy Days
Common terms:	What is 's surname on

The *StrIte* configuration begins with the conjunctive *AllTerms* query, and then iteratively drops terms until at least one matching sentence is returned. This deletion process selects firstly common terms, then name terms, and then finally quote terms, and within each of these groups selects terms in order of increasing informativeness (i.e. decreasing document frequency order). Using *StrIte* we find matching sentences for the question *What band did the music for the 1970's film "Saturday Night Fever"?*, which was not the case with previous strategies. This configuration produces answer sets for all questions.

Configuration *StrIteMorph* is a variant of *StrIte*, where each term is expanded with its morphological variant before querying the collection. As an example, the question *What city did Duke Ellington live in?* finds a matching sentence with the correct answer after removing term *city* and expanding term *live* with its morphological variant *lived*. Our final configuration *StrIteMorph20* is a variation of *StrIteMorph* in

Table 6: Strict coverage and redundancy for strategies *Allterms*, *WordNet*, *MorphVar*, *DropBig*, *DropSmall*, *BigIte*, *SmallIte*

	AllTerms		*WordNet*		*MorphVar*		*DropBig*		*DropSmall*		*BigIte*		*SmallIte*	
Rank	cov.	red.	cov.	red.	cov.	red.	cov.	red.	cov.	red.	cov.	red.	cov.	red.
1	6.0	0.06	6.6	0.07	8.0	0.08	9.3	0.09	7.1	0.07	11.8	0.12	7.1	0.07
5	12.7	0.19	12.9	0.19	14.9	0.21	19.6	0.28	15.4	0.22	25.4	0.35	15.7	0.22
10	14.6	0.25	14.9	0.26	16.8	0.28	22.3	0.36	17.4	0.29	30.6	0.50	17.6	0.30
20	16.8	0.32	17.1	0.32	19.3	0.35	25.1	0.46	19.3	0.35	34.5	0.64	20.1	0.36
30	19.0	0.38	19.0	0.38	21.5	0.41	28.1	0.53	21.8	0.42	38.6	0.74	22.3	0.43
50	20.7	0.45	20.9	0.45	23.4	0.48	30.6	0.61	24.3	0.49	41.9	0.85	24.8	0.51
100	22.1	0.54	22.3	0.54	24.8	0.57	32.6	0.71	26.2	0.60	45.0	0.98	27.6	0.62
200	22.6	0.60	22.9	0.60	25.1	0.62	33.7	0.78	27.0	0.67	46.9	1.09	28.1	0.69
500	22.9	0.71	23.2	0.71	25.4	0.74	34.2	0.91	27.9	0.79	47.5	1.24	29.0	0.83
1000	23.7	0.80	24.0	0.80	26.2	0.83	35.0	1.00	28.7	0.88	49.1	1.37	29.8	0.90

which the broadening technique is applied until *at least* 20 sentences per question are retrieved or no more relaxation can be done.

For these three configurations, we applied a different post-hoc ranking method, which differentially weights question terms Q in the quote, name and common groups, as in (3). For the experiments reported here, we stipulated a weight value $w(t)$ of 1/6 for common terms, 2/6 for name terms, and 3/6 for quote terms.

$$score(P, Q) = \sum_{t \in Q \cap P} w(t) * idf(t) \quad (3)$$

Coverage and redundancy figures for these three configurations are shown in Table 7. Comparing *StrIte* to *StrIteMorph*, we see that the effect of including morphological expansion is mixed, and mostly negative. Looking to the results for *StrIteMorph20*, we see that the impact of iterating until at least 20 matches are returned is clearly negative for ranks below twenty, but for higher ranks has a quite definitely positive effect. These results suggest that a promising line for investigation is iterative retrieval to some minimum number of results (here 20) combined with more effective re-ranking of results to get more answer bearing documents at lower ranks. A specific possibility suggested by comparing *StrIteMorph* and *StrIteMorph20* is that a ranking strategy that takes into account the "iteration step" in which a sentence was first found would lead to increased performance.

t-tests show that improvement in coverage is significative ($p < 0.01$) at all ranks between *StrIte* and *AllTerms*, *StrIteMorph* and *AllTerms*, and *StrIteMorph20* and *AllTerms*. While significative differences are only observed between *StrIteMorph20* and *StrIte* and bewteen *StrIteMorph20* and *StrIteMorph*.

5.5 Effect of Match Window and Passage Size

An additional issue we have investigated is whether improved coverage is gained by returning sentences additional to the one in which the query is matched, or by allowing the query to match in a window larger than just a single sentence. Initial results for allowing a larger match window with the *StrIteMorph* configuration have been negative, but this issue requires further investigation which we are pursuing, and whose results will be reported in due course.

Initial experiments using the *StrIteMorph20* configuration on matching within a single sentence but including additional adjacent sentences as part of the returned passage have produced positive results. When a sentence either side of the matching sentence is also returned (when they exist), the coverage of this configuration goes up to 60.50 at rank 100, and to 72.93 at rank 1000. The alternative of returning the two *preceding* sentences, produces somewhat less improvement: a coverage of 58.56 at rank 100 and 70.99 at rank 1000. As an example, the question *What is the state with the smallest population?* produces no answer bearing sentences under the standard *StrIteMorph20* configuration, but a sentence immediately preceding one of these matching sentences does provide the correct answer ("Wyoming"); other such cases exist. Note that these improved coverage results should be judged in light of the fact that a greater volume of text will be supplied to the answer extraction component, which may in fact, as a consequence, return less correct answers overall, due to the distraction effect of additional irrelevant material.

Table 7: Strict coverage and redundancy for strategies *StrIte*, *StrIteMorph*, and *StrIteMorph20*

	StrIte		*StrIteMorph*		*StrIteMorph20*	
Rank	cov.	red.	cov.	red.	cov.	red.
1	14.6	0.15	13.8	0.14	9.3	0.09
5	28.4	0.41	28.4	0.39	22.1	0.31
10	35.0	0.57	33.9	0.59	30.1	0.51
20	39.5	0.71	38.1	0.73	38.9	0.77
30	41.9	0.79	41.7	0.81	44.7	0.94
50	45.0	0.89	43.6	0.91	50.2	1.18
100	47.7	1.02	47.5	1.07	56.9	1.49
200	50.0	1.16	50.0	1.17	62.1	1.76
500	51.1	1.29	51.3	1.33	65.1	2.15
1000	51.3	1.40	53.0	1.49	68.5	2.42

6. DISCUSSION AND FUTURE WORK

In the foregoing we have explored various strategies by which natural language questions can be converted into boolean queries and used in conjunction with a boolean retrieval system and a post-retrieval ranking function to generate ranked passage sets for question answering. The goal of this work was to gain a deeper understanding of how more answer bearing passages could be returned at higher ranks, and in smaller passages so as to maximize the number of questions an answer extraction system is likely to be able to answer while minimizing the chance of distracting it with irrelevant text.

A number of valuable results and useful insights have emerged

from this work. First, while the numerically highest coverage and redundancy results we have been able to achieve so far using the boolean strategies described above are still not as good as the best results achieved by conventional ranked retrieval approaches, the story here is not entirely negative. At rank 200 StrIteMorph20 achieves 62.15% coverage, as compared to 72.9% for Lucene, 78.2% for Okapi, and 80.4% for Z-PRISE. However, if the amount of text each system returns at this rank is taken into account then we observe a direct correlation with coverage: StrIteMorph20 returns on average around 137 sentences at rank 200, Lucene around 300, Okapi around 800 and Z-PRISE around 4600. Choice of which of these approaches will lead to overall best system performance at question answering will depend on the ability of any downstream answer extraction component to avoid distraction in larger text volume.

Some avenues can be eliminated as simply not promising. For example, the simple minded extension of the basic *All-Terms* approach by using WordNet to expand all terms by all possible synonyms offers negligible advantage. Again, expanding all terms with morphological variants offers some advantage, but does not offer a major improvement.

However, most importantly, since the work carried out so far is by no means the end of the story, a number of promising avenues of research have been identified as being worthy of immediate pursuit:

- The post-retrieval ranking of results needs to be explored in more detail. For example, the weightings adopted for proper nouns, quoted phrases and other terms in our later experiments have no empirical basis and so optimal settings should be determined. Indeed, it remains to be verified whether the more complicated ranking scheme used for these experiments is in fact superior to the simpler method used for the earlier experiments. Other ranking methods should also be explored, such as simple overlap with the question (used, for example, in the MURAX system), and methods that use the stage during iterative query refinement that a particular result was first retrieved.
- The most effective window size within which to match boolean queries deserves further investigation, as does the potential benefits of returning additional sentences adjacent to the match window. This matter interacts with the further issue of the relation between the volume of text provided to an answer extraction system and that system's performance in correctly identifying answers. If we wish to limit the amount of text analysed, it is better on balance to go further down a ranked set of narrower passages, or to provide a smaller number of wider passages.
- The appropriate way to limit iterative query refinement should be further explored, for example whether it is better to have a larger or smaller lower limit for the number of results returned as the condition for stopping this iteration. There is also the question of whether we should apply an *upper* limit on the iteration, i.e. since if a very large number of results are returned for a boolean query, this may indicate that it has been weakened to the extent of no longer selecting useful passages.
- The impact of various possible term expansion methods deserves further attention. Although our preliminary results on using synonym expansion indicated only a small improvement in performance, these results are tied to a particular strategy, and so cannot be assumed to be more generally correct. Generalising queries by term expansion has obvious advantages over generalisation by the deletion of terms, i.e. the latter can more easily go too far and end up returning large number of unhelpful passages.

Work on some of these topics is currently underway. While addressing these topics is certainly not guaranteed to lead to coverage or redundancy results superior to those of more conventional ranked retrieval systems, these investigations should eventually lead to a better understanding of how to tailor retrieval and question manipulation for QA.

7. REFERENCES

[1] D. Dimmick. Guide to Z39.50/PRISE 2.0: Its Installation, Use, & Modification. `http://www.itl.nist.gov/iaui/894.02/works/papers/zp2/zp2.html`, 2000.

[2] R. Gaizauskas, M. A. Greenwood, M. Hepple, I. Roberts, H. Saggion, and M. Sargaison. The University of Sheffield's TREC 2003 Q&A Experiments. In *Proceedings of the 12th Text REtrieval Conference*, 2003.

[3] S. Harabagiu, D. Moldovan, M. Paşca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Gîrju, V. Rus, and P. Morărescu. FALCON: Boosting Knowledge for Answer Engines. In *Proceedings of the 9th Text REtrieval Conference*, 2000.

[4] J. Kupiec. MURAX: A Robust Linguistic Approach for Question Answering Using an On-Line Encyclopedia. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 181–190, 1993.

[5] M. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.

[6] I. Roberts and R. Gaizauskas. Evaluating passage retrieval approaches for question answering. In *Advances in Information Retrieval: Proceedings of the 26th European Conference on Information Retrieval (ECIR04)*, number 2997 in LNCS, pages 72–84, Sunderland, 2004. Springer.

[7] S. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Proceedings of the 8th Text REtrieval Conference*, 1999.

[8] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of the 3rd Text REtrieval Conference*, 1994.

[9] E. M. Voorhees. Overview of the TREC 2003 Question Answering Track. In *Proceedings of the 12th Text REtrieval Conference*, 2002.