Research Memorandum CS-03-06:

Evaluating Passage Retrieval Approaches for Question Answering

Ian Roberts and Robert Gaizauskas
Department of Computer Science, University of Sheffield, UK
{i.roberts,r.gaizauskas}@dcs.shef.ac.uk

Abstract

Automatic open domain question answering (QA) has been the focus of much recent research, stimulated by the introduction of a QA track in TREC in 1999. Many QA systems have been developed and most follow the same broad pattern of operation: first an information retrieval (IR) system, often passage-based, is used to find passages from a large document collection which are likely to contain answers, and then these passages are analysed in detail to extract answers from them. Most research to date has focused on this second stage, with relatively little detailed investigation into aspects of IR component performance which impact on overall QA system performance. In this paper, we (a) introduce two new metrics, *coverage* and *answer redundancy*, which we believe capture aspects of IR performance specifically relevant to QA more appropriately than do the traditional recall and precision measures, and (b) demonstrate their use in evaluating a variety of passage retrieval approaches using questions from TREC-9 and TREC 2001.

1 Introduction

The question answering (QA) evaluations in the Text REtrieval Conferences of 1999–2002, have encouraged wide interest in the QA problem. A review of the proceedings papers for TREC 2001 (in particular [10]) shows that the majority of systems entered in this evaluation operate using a broadly similar architecture. First, an information retrieval (IR) system is used to retrieve those documents or passages from the full collection which are believed to contain an answer to the question. In many cases, the question words are simply used as-is to form the retrieval system query, though a few systems make use of more advanced query processing techniques. Then, these retrieved passages are subjected to more detailed analysis, which may involve pattern matching or linguistic processing, in order to extract an answer to the question. The main differences between the competing systems lie in the details of this second stage.

There are several reasons for this two-stage architecture. Foremost is the relative efficiency of IR systems in comparison with the more complex and unoptimized natural language processing (NLP) techniques used in answer extraction. Most answer extraction components of QA systems simply could not be run in any reasonable time over document collections of the size of the TREC collection. This is due in part to more extensive processing, which may include part-of-speech tagging or semantic tagging or shallow parsing, but also because few NLP researchers have spent time separating "index-time" from "search-time" functionality and devising data structures to optimize the latter (though see [5, 6] for exceptions). Another reason for separating retrieval and answer extraction is that IR researchers have spent decades designing systems to achieve the best possible performance, in terms of precision and recall, in returning relevant documents from large text collections. To most NLP researchers it has seemed self-evident that one should take advantage of this work.

Given this two-stage architecture, most of the attention of the QA community has focused on the answer extraction component of QA systems. The first stage IR component is simply treated as a

black-box and relatively little work has been done to investigate in detail the effect that the quality of the IR stage has on systems' performance. Clearly, however, the second stage process can only determine an answer to a question if the passages retrieved by the first stage contain the necessary information. Furthermore, since, as Light et. al. [4] have shown, question answering systems tend to perform better for questions for which there are multiple answer occurrences in the document collection, an IR component that returns many occurrences of the answer in its top ranked documents is likely to be of more use in a QA system than one which returns few. QA systems such as the one developed by Microsoft Research [1] exploit this effect by searching for answers on the Web, where there is much greater answer redundancy than in the (relatively) small TREC collection.

In this paper, we concentrate on analyzing the performance of several different approaches to the information retrieval stage of QA, using metrics which aim to capture aspects of performance relevant to question answering. More specifically, we concentrate on investigating several different approaches to passage retrieval (PR). For a typical TREC question, such as *Where is the Taj Mahal?*, only a small section of a document will be required to determine the answer. Indeed, supplying a QA system with the full text of the document may in fact be counter-productive, as there will be many more opportunities for the system to become distracted from the correct answer by surrounding "noise". Therefore using an IR stage which supplies the QA system with limited-length "best" passages is an approach which many QA researchers have adopted, and is the approach we investigate here. Given significant variation in document length across the TREC collection, passage retrieval approaches have the additional benefit of permitting processing-bound answer extraction components to examine passages further down the passage ranking than would be possible were full documents to be used.

Deciding to adopt a passage retrieval approach, as opposed to a document retrieval approach, is still indeterminate in several regards. Different approaches to passage retrieval assume different notions of passage. Well-known distinctions [2] are between semantic, discourse and window-based notions of passage, in which passage boundaries are seen as marked by topic shifts, discourse markers, such as paragraph indicators, or fixed byte spans, respectively. Furthermore, regardless of which notion of passage one adopts, a number of additional choices must be made in deciding how best to implement passage retrieval for QA. For instance, do we divide documents into passages prior to indexing, and make the passage the unit of retrieval, or dynamically at search time after ranking the document collection overall? These two approaches might lead to significantly different rankings of the same passages, and this difference could have important implications for QA.

In the following paper we investigate a number of different approaches to passage retrieval for QA using two new metrics 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. This work is by no means exhaustive in terms of the PR approaches considered, and does not aim to be. Its central contribution is to introduce measures by which one can assess passage retrieval for question answering and to initiate debate about which approaches to PR may be best for QA.

2 Metrics for evaluating IR performance for QA

In the context of the QA task, the traditional IR performance measures of recall and precision demonstrate shortcomings that prompt us to define two new measures.

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:

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

The *answer redundancy* of a retrieval system S for a question set Q and document collection D at rank n is defined as:

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

The coverage gives the proportion of the question set for which a correct answer can be found within the top n passages retrieved for each question. The answer redundancy gives the average number, per question, of passages within the top n ranks retrieved which contain a correct answer.

In this framework, *precision* is defined as:

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

and *recall* as:

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

That is, the precision of a system for a given question set and document collection at rank n is the average proportion of the n returned documents or passages that contain a correct answer. Recall is the average proportion of answer bearing documents that are present in the top n returned documents or passages. In a QA context these global measures are not as helpful as coverage and redundancy. For example, suppose $n = 100$ and $|Q| = 100$. An IR system S_1 returning passages containing 100 correct answers in the top 100 ranks for a single question in $|Q|$ but 0 correct answers for all other questions receives the same precision score as a system S_2 returning exactly one correct answer bearing passage for each of the 100 questions in $|Q|$. However, S_1 when coupled to an answer extraction component of a QA system could answer at most one question correctly, while the S_2-based system could potentially answer all 100 questions correctly. Precision cannot capture this distinction, which is crucial for QA; coverage, on the other hand, captures exactly this distinction, in this case giving S_1 a score of 1 and S_2 a score of 100.

Recall is not as unhelpful as precision, and indeed one could argue that is more useful than redundancy as a measure, because it reveals to what extent the returned document set approaches the maximum redundancy obtainable, i.e. the extent to which all possible answering bearing passages are being returned. Redundancy, on the other hand, tells one only how many answering bearing passages per question are being returned on average. However, redundancy gives a neat measure of how many chances per question on average an answer extraction component has to find an answer, which is intuitively of interest in QA system development. More importantly, redundancy, being an absolute measure, can be compared across question and documents sets to give a measure of how difficult a specific QA task is. Furthermore, what answer redundancy misses, as compared to recall, can easily be captured by defining a notion of *actual redundancy* as $\sum_{q \in Q} |A_{D,q}|/|Q|$. This is the maximum answer redundancy any system could achieve. Comparing answer redundancy with actual redundancy captures the information that recall supplies, while giving overall information about the nature of the challenge presented by a specific question and document set which recall does not capture.

To obtain values for any of these measures, we must first decide what it means for an answer to be correct. In TREC, an answer is *correct* if it is a valid response to the question *and* if the document from which the answer is drawn provides evidence for the answer. This reflects the fact that an average user of a QA system does not trust the system absolutely, so an answer would only be accepted by the user if they could, in principle, verify it by reference to the original document. A candidate answer which is a valid response to the question, but which could not have been determined from the source document, is considered *unsupported*. Any other candidate answer is considered *incorrect*. The judgment of an answer's correctness or otherwise is determined by a human assessor.

While this kind of manual evaluation is feasible for a one-off evaluation such as TREC, a similar process is not reasonable for repeated experiments on the retrieval system. An assessor would have to examine every passage retrieved to determine whether it (a) contained an answer to the question and (b) supported that answer. With potentially hundreds of passages to examine per question and hundreds of questions in the test set, this adds up to several hundred thousand passages per run. Also, since human judgments are inherently subjective, the same set of answers to the same questions, based on the same documents, will be scored differently by different assessors, so the results will not be repeatable. Clearly, an automatic method of assessment is needed.

Voorhees and Tice [11] describe a possible solution to this problem. For the TREC collection,

NIST have created regular expression patterns, intended to match strings which answer each question, and a set of relevance judgments, assembled from the combined results of all participating systems, that indicate which documents provide supporting evidence for answers to each question. For our purposes, a passage is considered to contain a correct answer to a question if some substring of the passage matches one of the regular expressions for the question, and the document from which the passage was drawn is judged to be relevant.[1]

3 Alternative Approaches to Passage Retrieval

For the TREC-9 QA track our QA system [8], which adopts the two stage model for QA introduced in section 1, employed Okapi [7] as the IR component. For the reasons outlined in section 1 we wanted to use a passage-based approach and so relied upon Okapi's native support for paragraph-based passage retrieval. While using the native passage retrieval support of an IR engine such as Okapi was convenient, we became aware that the technique used by the engine might not be the most suitable for the question answering application. For example, Okapi will never retrieve more than one passage from the same source document, though it is quite possible that several such passages may be relevant to the question. There are essentially two ways to address this issue:

1. Pre-process the document collection, breaking documents into their component passages before indexing. The retrieval system then treats each passage as a document in its own right.
2. Retrieve full documents from the retrieval system, then break each document into its component passages and perform a second retrieval run to find the best passages across the retrieved document set.

With this context in mind, and keeping open the possibility that full document-based ranking may be superior to passage-based ranking, we investigated five approaches to passage retrieval:

Okapi According to [7], Okapi's native approach to passage retrieval 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. Given a query the retrieval engine first treats each document as a single passage and considers all documents whose weight exceeds a threshold set empirically at the weight of the 10,000th document. The documents above 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.

Approach 1 In this approach, all documents are pre-processed to produce a new document collection consisting of all passages drawn from the original document set which are then indexed. For consistency with Okapi, we again use paragraphs as passages. At search time the best passages are returned, possibly several from each document, in the order determined by the document ranking algorithm.

Approach 2 In this approach the top n retrieved full documents are post-processed into passages. For the ith retrieved document ($i = 1, 2 \ldots n$), a document collection is built from its passages, and a second stage retrieval is run against this collection, using the same retrieval engine as in the first stage, to determine the best passage from that document. The text of this passage is then returned as the ith passage in the final ranking. Thus, full document retrieval is assumed to get the overall ranking right, but only the best passage from each document is selected for further processing.

Approach 3 In this approach, the top n retrieved full documents are again post-processed into passages, but this time, a single second-stage index is built from the passages from all n documents. The best n passages are then selected from this index, using the same retrieval engine as in the first stage, allowing multiple passages per document to be returned.

[1] If a question has multiple possible answers, it is possible that the passage contains one of these answers, but the document from which the passage was drawn supports a different answer, but we believe such situations to be sufficiently rare that they will not be considered further.

Approach 4 This approach is like approach 3, except that the second retrieval stage is limited to retrieve at most one passage from each original document. Thus, only one passage per document is returned, as in approach 2, but the ranking is determined by the passage score rather than the full document score. This simulates the Okapi approach, and is included primarily as a control, as non-Okapi-based tools were used to implement approaches 1–4 (see next section).

Thus, to summarize, only approach 1 does index-time passaging, the other four approaches do search-time passaging. The differences between them are to do with whether the original ranking resulting from the initial query should guide the subsequent passage ranking (approach 2) or not (approaches 3 and 4 and Okapi) and whether one passage per document (Okapi, approaches 2 and 4) or multiple passages per document (approaches 1 and 3) should be returned.

Clearly these variations do not exhaust the space of possible approaches to passage retrieval. However, they provide an initial set to explore to see if significant differences in results begin to emerge.

4 Implementation

To run Okapi we simply downloaded the publicly available version [2] and used it as is.

To investigate the other approaches we used Lemur[3] as the underlying retrieval engine. Lemur has native support for the TREC document format, and supports vector-space, probabilistic and language modelling retrieval approaches against a single index. To keep experiments with Lemur as comparable as possible to those with Okapi we report here only the results of using the probabilistic approach (BM25 term weighting) within Lemur, as this is the model used in Okapi. We did investigate the other approaches supported by Lemur, but these had little significant effect.

To carry out passaging, a Perl program was written to read the source documents and split them into passages one paragraph in length. The line offsets of the passages within the original source files are stored in a flat index file to enable the passages to be reconstructed from the original data. The passages are output as TREC-formatted documents, which are then passed to Lemur for indexing. The index is built using the same list of stopwords as for Okapi.

There are some important practical issues of scalability that distinguish the pre-processing passaging approach (approach 1) from those that do passaging after initial retrieval. By treating every passage as an individual document, the pre-processing approach vastly increases both the space and time requirements of the indexing and retrieval programs. The subset of the TREC collection we used for testing (see next section) consists of 242,918 separate documents, and the full-document index built for the post-processing approaches required 1,122MB of disk space. The retrieval program took about 40 seconds to load the index, and at its peak, it consumed about 50MB of memory.[4]

In comparison, the preprocessor generated over 3.7 million separate passages (each a separate document to the IR system), a 15 fold increase. Though the index required only 1.4GB to store, the larger number of smaller documents meant that the retrieval program consumed 300MB of memory and took several minutes of intensive processing to load the index. In addition, the passage location index, used to map passage IDs back into the source text, required 130MB of space.

5 Experiments and Results

The test set used for these experiments was derived from the combined set of 1193 questions from TREC-9 (2000) and TREC 2001. These two evaluations operated over the same document set, and relevance judgments and answer patterns are available for both evaluations from NIST. The documents in the TREC collection are sourced from a variety of newswires by NIST, including the Associated Press (AP) newswire, the Wall Street Journal, the Los Angeles Times and the San José Mercury News. The documents are marked up in SGML, and the format of the markup varies from source to source. In particular, an algorithm to split documents into paragraphs for one source will not work

[2] `http://www.soi.city.ac.uk/~andym/OKAPI-PACK`

[3] `http://www-2.cs.cmu.edu/~lemur/`

[4] These results were obtained on a dual processor UltraSPARC, running Solaris 8, with 2GB of main memory.

	% coverage at rank						
Run type	5	10	20	30	50	100	200
Okapi	48.78	60.02	66.63	69.76	74.51	78.79	82.04
Approach 1	43.80	54.58	63.50	67.67	71.38	78.22	83.43
Approach 2	45.89	55.39	63.73	67.21	72.31	76.25	79.72
Approach 3	41.48	54.11	63.85	68.48	73.70	80.07	85.52
Approach 4	40.79	52.26	60.37	65.47	69.06	74.39	77.87

Table 1: Results of passage retrieval experiments – coverage

	Answer redundancy at rank						
Run type	5	10	20	30	50	100	200
Okapi	0.877	1.414	1.919	2.226	2.644	3.118	3.426
Approach 1	0.761	1.255	1.833	2.196	2.679	3.488	4.216
Approach 2	0.771	1.171	1.657	1.933	2.312	2.773	3.126
Approach 3	0.729	1.200	1.831	2.251	2.862	3.808	4.757
Approach 4	0.706	1.127	1.607	1.906	2.312	2.752	3.017

Table 2: Results of passage retrieval experiments – answer redundancy

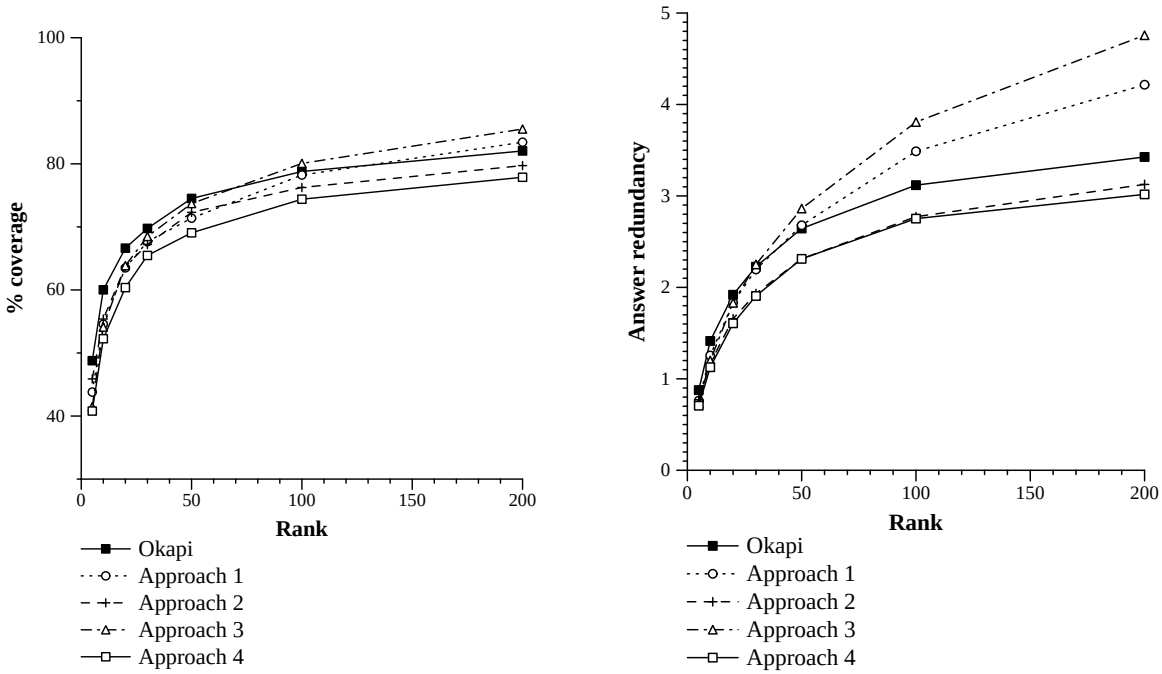

Figure 1: Results of passage retrieval experiments

for any of the other documents. In view of this, the experiments detailed below are based only on documents from one source. We chose the AP newswire, as 72% (863) of the 1193 test questions have at least one relevant document from this collection (i.e. a *correct* judgment with an AP document as the justification). The "next best" collection, in this sense, is the Los Angeles Times, for which only 53% of the questions have a relevant document. [5]

Each of the five approaches was evaluated by using each question in the question set as a query and returning the top 200 passages. For approaches which involved a two step process using Lemur (approaches 2, 3 and 4), 200 documents were retrieved in step one, then passaging was carried out and 200 passages were returned in step two in the manner of the specific approach. [6]

To inform our analysis of the results, we also calculated the actual redundancy, as defined in section 2, as follows. For each question we used the human assessors' judgments to pull out from the AP collection the documents identified as relevant to that question. For each of these documents we then split it into paragraphs, tested each of the NIST-supplied Perl patterns against each paragraph, and counted how many paragraphs matched at least one pattern. Actual redundancy for each question is then the total of these counts over all documents identified as answer bearing. Overall actual redundancy is the average of these redundancies per question. Note that this is still an estimate (a lower bound) of true redundancy because the assessors only confirm those documents as containing answers if they have been proposed by some system. Using this approach we determined that the actual redundancy is 14.3. This is the highest answer redundancy score a system could achieve under our scoring system, if it retrieved every pattern-matching paragraph from every relevant document in the AP collection.

Tables 1 and 2, and the corresponding figure 1, show the results of the experiments. We see that the best coverage is consistently obtained by the native Okapi passage retrieval mechanism at the highest ranks, but that it is gradually overtaken by approaches 1 and 3 at lower ranks, though only marginally. However, Okapi has considerably lower answer redundancy scores than approaches 1 and 3. The most likely cause of this is that Okapi is limited to retrieving no more than one passage per document, and it seems highly likely, though we have not assembled the data to prove it, that in many cases the multiple answer instances contributing to the 14.3 actual redundancy figure fall within different passages in the same document. This conjecture is supported by the poor redundancy scores of approaches 2 and 4, which are also limited to returning only one passage per document. By contrast, those approaches which are able to retrieve multiple passages from the same document (approaches 1 and 3) demonstrate higher answer redundancy at all but the highest ranks. For the two approaches which can retrieve multiple passages per document, an examination of the average number of passages per source document retrieved per question reveals that approach 1 retrieves on average 1.2 passages per document, whereas approach 3 retrieves 1.6 – 33% more passages per document on average than approach 1.

Overall combined best performance in terms of coverage and answer redundancy, including passages down to rank 200, is obtained by approach 3. This is agreeable since approach 3 does not suffer from the space and time efficiency problems affecting approach 1. Of course while the approaches seem to be diverging at rank 200, we cannot rule out the possibility of their relative positions changing at even lower ranks.

6 Conclusions

We have investigated five approaches to paragraph-based passage retrieval for question answering, varying principally as to whether:

- they divide documents into passages prior to indexing, effectively treating each passage as an independent document, or after an initial retrieval stage;
- they permit only one or more than one passage per document to be returned;

[5] There is some evidence that the AP newswire documents may not be representative of the whole collection – see [3].

[6] We restricted the maximum number of passages returned to 200 due to system limitations. Our QA system requires each retrieved passage to be stored as a separate file in the file system and the effect of moving beyond rank 200 (for 863 qyestions and five runs) was to run out of i-nodes on our Unix server. A more efficient representation is required to extend these experiments to lower ranks.

- for search-time passaging approaches, the final passage ranking should be guided by the ranking of full documents resulting from the initial query or by the ranking obtained in the secondary passage retrieval stage.

To evaluate the utility of these approaches for question answering we have introduced two new metrics, *coverage* and *answer redundancy* which capture what proportion of the question set has at least one answer returned in the top n passages and the average number of repetitions of the answer in the top n passages, respectively. These metrics, we believe, are intuitive measures of the suitability of a passage retrieval approach for QA.

Applying these metrics to assess five approaches to passage retrieval in one specific experiment using TREC QA data, we determined that the best-performing passage retrieval approach was one that first does full document retrieval, then splits the top retrieved documents into passages and performs a second passage retrieval operation against this passage set, returning the passages in the rank order determined by the second retrieval operation. This approach obtains both better coverage and answer redundancy scores beyond about rank 100.

A number of further questions immediately suggest themselves. Our experiment was restricted to the top 200 ranks. While the scores for the approaches appear to be diverging at this point, further experimentation at lower ranks should be carried out to confirm this. Of particular interest are the points at which coverage reaches 100% and answer redundancy approaches actual redundancy.

One would like to see higher coverage and redundancy at higher ranks. Can this be achieved using other passage retrieval approaches not explored here? Or, are current performance levels unsurpassable, given an approach which uses the raw question words as the query to the retrieval system? Various approaches to query "enhancement" need to be considered.

While higher coverage and answer redundancy would appear to be inherently good for QA systems, there may a critical tradeoff between specific values for coverage and redundancy and the rank at which are these are obtained. For example, a QA system may do better with the top 50 passages than with the top 100, even though the top 100 have higher coverage and redundancy, simply because of the "noise" introduced by a further 50 passages. The interaction between coverage and redundancy at certain ranks and the answer extraction capabilities of QA systems needs to be investigated.

References

[1] E. Brill, J. Lin, M. Banko, S. Dumais, and A. Ng. Data-intensive question answering. In [9], pp. 393–400.

[2] J. P. Callan. Passage-level evidence in document retrieval. In *Proc. of the 17th ACM SIGIR conference*, pp. 302-310, 1994.

[3] W. B. Cavnar. N-gram based text filtering for TREC-2. In *NIST Special Publication 500-215: The Second Text REtrieval Conference (TREC-2)*, pp. 171–179, 1993.

[4] M. Light, G. S. Mann, E. Riloff, and E. Breck. Analyses for elucidating current question answering technology. *Natural Language Engineering*, 7(4):325–342, 2001.

[5] D. Milward and J. Thomas. From information retrieval to information extraction. In *Proc. of the ACL Workshop on Recent Advances in Natural Language Processing and Information Retrieval*, 2000. Available at: http://www.cam.sri.com/html/highlight.html.

[6] D. Molla Aliod, J. Berri, and M. Hess. A real world implementation of answer extraction. In *Proc. of the 9th Int. Conference on Database and Expert Systems Applications Workshop "Natural Language and Information Systems" (NLIS'98)*, pp. 143–148, 1998.

[7] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *NIST Special Publication 500-225: The Third Text REtrieval Conference (TREC-3)*, pp. 109–126, 1994.

[8] S. Scott and R. Gaizauskas. University of Sheffield TREC-9 Q & A System. In *Proc. of The Ninth Text REtrieval Conference (TREC 9)*, pp. 635–644. NIST Special Publication 500-249, 2000.

[9] E. Voorhees and D. Harman, editors. *NIST Special Publication 500-250: The Tenth Text REtrieval Conference (TREC 2001)*, 2001.

[10] E. M. Voorhees. Overview of the TREC-2001 question answering track. In [9].

[11] E. M. Voorhees and D. M. Tice. Building a question answering test collection. In *Proc. of the 23rd ACM SIGIR conference*, pp. 200–207, 2000.