In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002), August, 2002

Web Question Answering: Is More Always Better?

Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, Andrew Ng
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
{sdumais, mbanko, brill}@research.microsoft.com
jimmylin@ai.mit.edu; ang@cs.berkeley.edu

ABSTRACT

This paper describes a question answering system that is designed to capitalize on the tremendous amount of data that is now available online. Most question answering systems use a wide variety of linguistic resources. We focus instead on the redundancy available in large corpora as an important resource. We use this redundancy to simplify the query rewrites that we need to use, and to support answer mining from returned snippets. Our system performs quite well given the simplicity of the techniques being utilized. Experimental results show that question answering accuracy can be greatly improved by analyzing more and more matching passages. Simple passage ranking and n-gram extraction techniques work well in our system making it efficient to use with many backend retrieval engines.

Categories and Subject Descriptors

H.3.1. [**Content Analysis and Indexing**], H.3.3 [**Information Search and Retrieval**].

General Terms

Algorithms, Experimentation.

1. INTRODUCTION

Question answering has recently received attention from the information retrieval, information extraction, machine learning, and natural language processing communities [1][3][19][20] The goal of a question answering system is to retrieve 'answers' to questions rather than full documents or even best-matching passages as most information retrieval systems currently do. The TREC Question Answering Track which has motivated much of the recent work in the field focuses on fact-based, short-answer questions such as "*Who killed Abraham Lincoln?*" or "*How tall is Mount Everest?*". In this paper we focus on this kind of question answering task, although the techniques we propose are more broadly applicable.

The design of our question answering system is motivated by recent observations in natural language processing that, for many applications, significant improvements in accuracy can be attained simply by increasing the amount of data used for learning. Following the same guiding principle we take advantage of the tremendous data resource that the Web provides as the backbone of our question answering system. Many groups working on question answering have used a variety of linguistic resources – part-of-speech tagging, syntactic parsing, semantic relations, named entity extraction, dictionaries, WordNet, etc. (e.g., [2][8][11][12][13][15][16]).We chose instead to focus on the Web as gigantic data repository with tremendous redundancy that can be exploited for question answering. The Web, which is home to billions of pages of electronic text, is orders of magnitude larger than the TREC QA document collection, which consists of fewer than 1 million documents. This is a resource that can be usefully exploited for question answering. We view our approach as complimentary to more linguistic approaches, but have chosen to see how far we can get initially by focusing on data per se as a key resource available to drive our system design.

Automatic QA from a single, small information source is extremely challenging, since there is likely to be only one answer in the source to any user's question. Given a source, such as the TREC corpus, that contains only a relatively small number of formulations of answers to a query, we may be faced with the difficult task of mapping questions to answers by way of uncovering complex lexical, syntactic, or semantic relationships between question string and answer string. The need for anaphor resolution and synonymy, the presence of alternate syntactic formulations, and indirect answers all make answer finding a potentially challenging task. However, the greater the answer redundancy in the source data collection, the more likely it is that we can find an answer that occurs in a simple relation to the question. Therefore, the less likely it is that we will need to resort to solving the aforementioned difficulties facing natural language processing systems.

2. EXPLOITING REDUNDANCY FOR QA

We take advantage of the redundancy (multiple, differently phrased, answer occurrences) available when considering massive amounts of data in two key ways in our system.

Enables Simple Query Rewrites. The greater the number of information sources we can draw from, the easier the task of rewriting the question becomes, since the answer is more likely to be expressed in different manners. For example, consider the difficulty of gleaning an answer to the question *"Who killed Abraham Lincoln?"* from a source which contains only the text *"John Wilkes Booth altered history with a bullet. He will forever be known as the man who ended Abraham Lincoln's life,"*

SIGIR'02, August 11-15, 2002, Tampere, Finland.

versus a source that also contains the transparent answer string, *"John Wilkes Booth killed Abraham Lincoln."*

Facilitates Answer Mining. Even when no obvious answer strings can be found in the text, redundancy can improve the efficacy of question answering. For instance, consider the question: "*How many times did Bjorn Borg win Wimbledon?*" Assume the system is unable to find any obvious answer strings, but does find the following sentences containing "Bjorn Borg" and "Wimbledon", as well as a number:

(1) **Bjorn Borg** blah blah **Wimbledon** blah blah **5** blah

(2) **Wimbledon** blah blah blah **Bjorn Borg** blah **37** blah.

(3) blah **Bjorn Borg** blah blah **5** blah blah **Wimbledon**

(4) **5** blah blah **Wimbledon** blah blah **Bjorn Borg**.

By virtue of the fact that the most frequent number in these sentences is 5, we can posit that as the most likely answer.

3. RELATED WORK

Other researchers have recently looked to the web as a resource for question answering. The Mulder system described by Kwok et al. [14] is similar to our approach in several respects. For each question, Mulder submits multiple queries to a web search engine and analyzes the results. Mulder does sophisticated parsing of the query and the full-text of retrieved pages, which is far more complex and compute-intensive than our analysis. They also require global idf term weights for answer extraction and selection, which requires local storage of a database of term weights. They have done some interesting user studies of the Mulder interface, but they have not evaluated it with TREC queries nor have they looked at the importance of various system components.

Clarke et al. [9][10] investigated the importance of redundancy in their question answering system. In [9] they found that the best weighting of passages for question answering involves using both passage frequency (what they call redundancy) and a global idf term weight. They also found that analyzing more top-ranked passages was helpful in some cases and not in others. Their system builds a full-content index of a document collection, in this case TREC. In [10] they use web data to reinforce the scores of promising candidate answers by providing additional redundancy, with good success. Their implementation requires an auxiliary web corpus be available for full-text analysis and global term weighting. In our work, the web is the primary source of redundancy and we operate without a full-text index of documents or a database of global term weights.

Buchholz's Shapaqa NLP system [7] has been evaluated on both TREC and Web collections. Question answering accuracy was higher with the Web collection (although both runs were poor in absolute terms), but few details about the nature of the differences are provided.

These systems typically perform complex parsing and entity extraction for both queries and best matching web pages ([7][14]), which limits the number of web pages that they can analyze in detail. Other systems require term weighting for selecting or ranking the best-matching passages ([10][14]) and this requires auxiliary data structures. Our approach is distinguished from these in its simplicity (simple rewrites and string matching) and efficiency in the use of web resources (use of only summaries and simple ranking). We now describe how our system uses redundancy in detail and evaluate this systematically.

4. SYSTEM OVERVIEW

A flow diagram of our system is shown in Figure 1. The system consists of four main components.

Rewrite Query. Given a question, the system generates a number of rewrite strings, which are likely substrings of declarative answers to the question. To give a simple example, from the question "*When was Abraham Lincoln born*?" we know that a likely answer formulation takes the form "Abraham Lincoln was born on <DATE>". Therefore, we can look through the collection of documents, searching for such a pattern.

We first classify the question into one of seven categories, each of which is mapped to a particular set of rewrite rules. Rewrite rule sets range in size from one to five rewrite types. The output of the rewrite module is a set of 3-tuples of the form [`string, L/R/-, weight`], where "`string`" is the reformulated

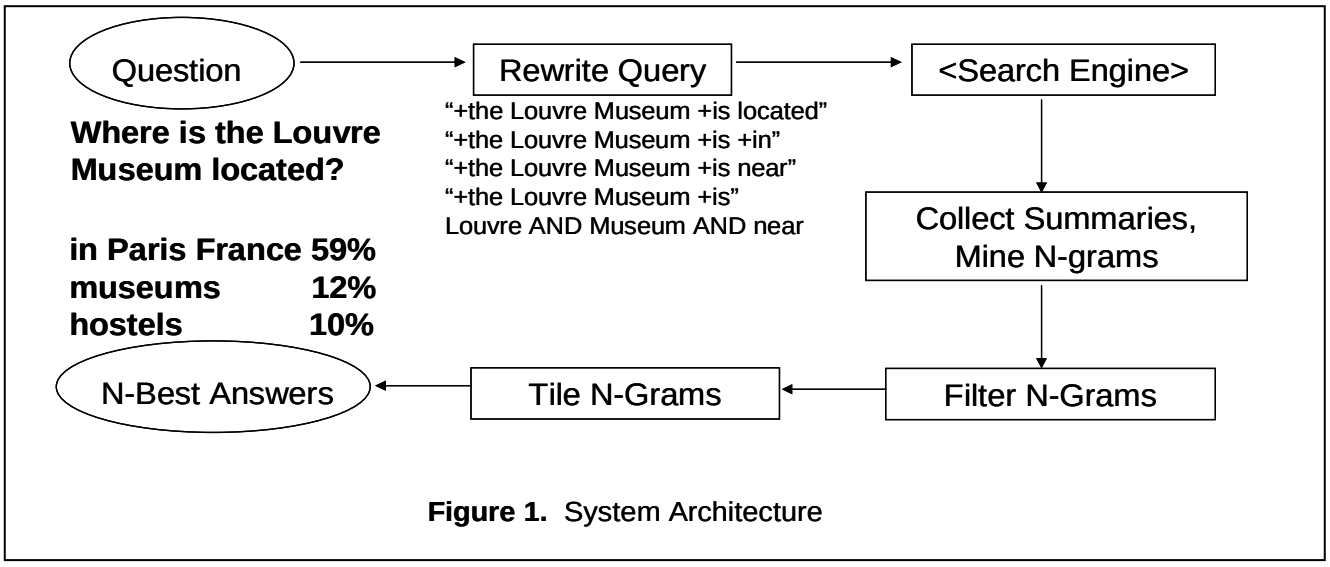

Figure 1. System Architecture

search query, "`L/R/-`" indicates the position in the text where we expect to find the answer with respect to the query string (to the left, right or anywhere) and "`weight`" reflects how much we prefer answers found with this particular query. The idea behind using a weight is that answers found using a high precision query (e.g., "Abraham Lincoln was born on") are more likely to be correct than those found using a lower precision query (e.g., "Abraham" AND "Lincoln" AND "born").

We do not use a parser or part-of-speech tagger for query reformulation, but do use a lexicon in order to determine the possible parts-of-speech of a word as well as its morphological variants. We created the rewrite rules and associated weights manually for the current system, although it may be possible to learn query-to-answer reformulations and weights (e.g. see Agichtein et al. [4]; Radev et al. [17]).

The rewrites generated by our system are simple string-based manipulations. For instance, some question types involve query rewrites with possible verb movement; the verb "*is*" in the question "*Where is the Louvre Museum located*?" should be moved in formulating the desired rewrite to "The Louvre Museum **is** located in". While we might be able to determine where to move a verb by analyzing the sentence syntactically, we took a much simpler approach. Given a query such as "Where is w_1 w_2 … w_n", where each of the w_i is a word, we generate a rewrite for each possible position the verb could be moved to (e.g. "w_1 is w_2 … w_n", "w_1 w_2 is … w_n", etc). While such an approach results in many nonsensical rewrites (e.g. "The Louvre is Museum located in"), these very rarely result in the retrieval of bad pages, and the proper movement position is guaranteed to be found via exhaustive search. If we instead relied on a parser, we would require fewer query rewrites, but a misparse would result in the proper rewrite not being found.

For each query we also generate a final rewrite which is a backoff to a simple ANDing of the non-stop words in the query. We could backoff even further to ranking using a best-match retrieval system which doesn't require the presence of all terms and uses differential term weights, but we did not find that this was necessary when using the Web as a source of data. There are an average of 6.7 rewrites for the 500 TREC-9 queries used in the experiments described below.

As an example, the rewrites for the query "*Who created the character of Scrooge?*" are:

> LEFT_5_"created +the character +of Scrooge"
>
> RIGHT_5_"+the character +of Scrooge +was created +by"
>
> AND_2_"created" AND "+the character" AND "+of Scrooge"
>
> AND_1_"created" AND "character" AND "Scrooge"

To date we have used only simple string matching techniques. Soubbotin and Soubbotin [18] have used much richer regular expression matching to provide hints about likely answers, with very good success in TREC 2001, and we could certainly incorporate some of these ideas in our rewrites. Note that many of our rewrites require the matching of stop words like "in" and "the", in the above example. In our system stop words are important indicators of likely answers, and we do not ignore them as most ranked retrieval systems do, except in the final backoff AND rewrite.

The query rewrites are then formulated as search engine queries and sent to a search engine from which page summaries are collected and analyzed.

Mine N-Grams. From the page summaries returned by the search engine, n-grams are mined. For reasons of efficiency, we use only the returned summaries and not the full-text of the corresponding web page. The returned summaries contain the query terms, usually with a few words of surrounding context. In some cases, this surrounding context has truncated the answer string, which may negatively impact results. The summary text is then processed to retrieve only strings to the left or right of the query string, as specified in the rewrite triple.

1-, 2-, and 3-grams are extracted from the summaries. An N-gram is scored according the weight of the query rewrite that retrieved it. These scores are summed across the summaries that contain the n-gram (which is the opposite of the usual inverse document frequency component of document/passage ranking schemes). We do not count frequency of occurrence within a summary (the usual tf component in ranking schemes). Thus, the final score for an n-gram is based on the rewrite rules that generated it and the number of unique summaries in which it occurred. When searching for candidate answers, we enforce the constraint that at most one stopword is permitted to appear in any potential n-gram answers.

The top-ranked n-grams for the Scrooge query are:

> Dickens 117
> Christmas Carol 78
> Charles Dickens 75
> Disney 72
> Carl Banks 54
> A Christmas 41
> uncle 31

Filter/Reweight N-Grams. Next, the n-grams are filtered and reweighted according to how well each candidate matches the expected answer-type, as specified by a handful of handwritten filters. The system uses filtering in the following manner. First, the query is analyzed and assigned one of seven question types, such as *who-question*, *what-question*, or *how-many-question*. Based on the query type that has been assigned, the system determines what collection of filters to apply to the set of potential answers found during n-gram harvesting. The answers are analyzed for features relevant to the filters, and then rescored according to the presence of such information.

A collection of approximately 15 filters were developed based on human knowledge about question types and the domain from which their answers can be drawn. These filters used surface string features, such as capitalization or the presence of digits, and consisted of handcrafted regular expression patterns.

After the system has determined which filters to apply to a pool of candidate answers, the selected filters are applied to each candidate string and used to adjust the score of the string. In most cases, filters are used to boost the score of a potential answer when it has been determined to possess the features relevant to the query type. In other cases, filters are used to remove strings from the candidate list altogether. This type of exclusion was only performed when the set of correct answers was determined to be a

closed set (e.g. "Which continent….?") or definable by a set of closed properties (e.g. "How many…?").

Tile N-Grams. Finally, we applied an answer tiling algorithm, which both merges similar answers and assembles longer answers out of answer fragments. Tiling constructs longer n-grams from sequences of overlapping shorter n-grams. For example, "A B C" and "B C D" is tiled into "A B C D." The algorithm proceeds greedily from the top-scoring candidate - all subsequent candidates (up to a certain cutoff) are checked to see if they can be tiled with the current candidate answer. If so, the higher scoring candidate is replaced with the longer tiled n-gram, and the lower scoring candidate is removed. The algorithm stops only when no n-grams can be further tiled.

The top-ranked n-grams after tiling for the Scrooge query are:

Charles Dickens 117
A Christmas Carol 78
Walt Disney's uncle 72
Carl Banks 54
uncle 31

Our system works most efficiently and naturally with a backend retrieval system that returns best-matching passages or query-relevant document summaries. We can, of course, post-process the full text of matching documents to extract summaries for n-gram mining, but this is inefficient especially in Web applications where the full text of documents would have to be downloaded over the network at query time.

5. EXPERIMENTS

For our experimental evaluations we used the first 500 TREC-9 queries (201-700) [19]. For simplicity we ignored queries which are syntactic rewrites of earlier queries (701-893), although including them does not change the results in any substantive way. We used the patterns provided by NIST for automatic scoring. A few patterns were slightly modified to accommodate the fact that some of the answer strings returned using the Web were not available for judging in TREC-9. We did this in a very conservative manner allowing for more specific correct answers (e.g., Edward J. Smith vs. Edward Smith) but not more general ones (e.g., Smith vs. Edward Smith), and simple substitutions (e.g., 9 months vs. nine months). These changes influence the absolute scores somewhat but do not change relative performance, which is our focus here.

Many of the TREC queries are time sensitive – that is, the correct answer depends on when the question is asked. The TREC database covers a period of time more than 10 years ago; the Web is much more current. Because of this mismatch, many correct answers returned from the Web will be scored as incorrect using the TREC answer patterns. 10-20% of the TREC queries have temporal dependencies. E.g., *Who is the president of Bolivia? What is the exchange rate between England and the U. S.?* We did not modify the answer key to accommodate these time differences, because this is a subjective job and would make comparison with earlier TREC results impossible.

For the main Web retrieval experiments we used Google as a backend because it provides query-relevant summaries that make our n-gram mining techniques more efficient. Thus we have access to more than 2 billion web pages. For some experiments in TREC retrieval we use the standard QA collection consisting of news documents from Disks 1-5. The TREC collection has just under 1 million documents [19].

All runs are completely automatic, starting with queries and generating a ranked list of 5 candidate answers. Candidate answers are a maximum of 50 bytes long, and typically much shorter than that. We report the Mean Reciprocal Rank (MRR) of the first correct answer, the Number of Questions Correctly Answered (NumCorrect), and the Proportion of Questions Correctly Answered (PropCorrect). Correct answers at any rank are included in the number and proportion correct measures. Most correct answers are at the top of the list -- 70% of the correct answers occur in the first position and 90% in the first or second positions.

Using our system with default settings for query rewrite weights, number of summaries returned, etc. we obtain a MRR of 0.507 and answer 61% of the queries. The average answer length was 12 bytes, so the system is really returning short answers not passages. This is very good performance and would place us near the top of 50-byte runs for TREC-9. However, since we did not take part in TREC-9 it is impossible to compare our results precisely with those systems (e.g., we used TREC-9 for tuning our TREC-10 system increasing our score somewhat, but we return several correct answers that were not found in TREC-9 thus decreasing our score somewhat).

Redundancy is used in two key ways in our data-driven approach. First, the occurrence of multiple linguistic formulations of the same answers increases the chances of being able to find an answer that occurs within the context of a simple pattern match with the query. Second, answer redundancy facilitates the process of answer extraction by giving higher weight to answers that occur more often (i.e., in more different document summaries). We now evaluate the contributions of these experimentally.

5.1 Number of Snippets

We begin by examining the importance of redundancy in answer extraction. To do this we vary the number of summaries (snippets) that we get back from the search engine and use as input to the n-gram mining process. Our standard system uses 100 snippets. We varied the number of snippets from 1 to 1000. The results are shown in Figure 2.

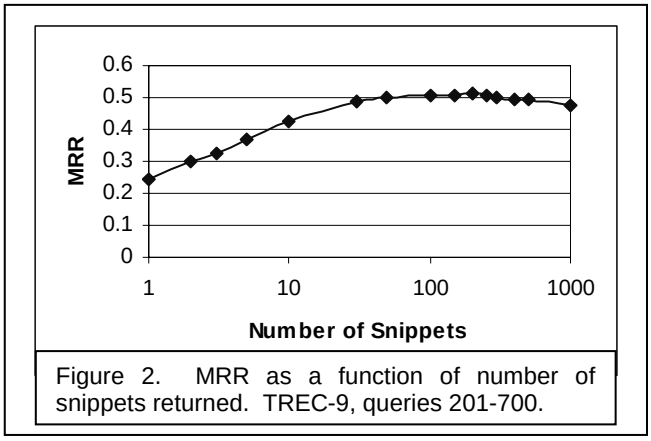

Figure 2. MRR as a function of number of snippets returned. TREC-9, queries 201-700.

Performance improves sharply as the number of snippets increases from 1 to 50 (0.243 MRR for 1 snippet, 0.370 MRR for 5, 0.423 MRR for 10, and 0.501 for 50), somewhat more slowly after that

(peaking 0.514 MRR with 200 snippets), and then falling off somewhat after that as more snippets are included for n-gram analysis. Thus, over quite a wide range, the more snippets we consider in selecting and ranking n-grams the better. We believe that the slight drop at the high end is due to the increasing influence that the weaker rewrites have when many snippets are returned. The most restrictive rewrites return only a few matching documents. Increasing the number of snippets increases the number of the least restrictive matches (the AND matches), thus swamping the restrictive matches. In addition, frequent n-grams begin to dominate our rankings at this point.

An example of failures resulting from too many AND matches is Query 594: *What is the longest word in the English language?* For this query, there are 40 snippets matching the rewrite "is the longest word in the English language" with weight 5, 40 more snippets matching the rewrite "the longest word in the English language is" with the weight 5, and more than 100 snippets matching the backoff AND query ("longest" AND "word" AND "English" AND "language") with a weight of 1. When 100 snippets are used, the precise rewrites contribute almost as many snippets as the AND rewrite. In this case we find the correct answer, "pneumonoultramicroscopicsilicovolcanokoniosis", in the second rank. The first answer, "1909 letters long", which is incorrect, also matches many precise rewrites such as "the longest word in English is ## letters long", and we pick up on this. When 1000 snippets are used, the weaker AND rewrites dominate the matches. In this case, the correct answer falls to seventh on the list after "letters long", "one syllable", "is screeched", "facts", "stewardesses" and "dictionary", all of which occur commonly in results from the least restrictive AND rewrite. A very common AND match contains the phrase "the longest one-syllable word in the English language is screeched", and this accounts for two of our incorrect answers.

Using differential term weighting of answer terms, as many retrieval systems do, should help overcome this problem to some extent but we would like to avoid maintaining a database of global term weights. Alternatively we could refine our weight accumulation scheme to dampen the effects of many weakly restrictive matches by sub-linear accumulation, and we are currently exploring several alternatives for doing this.

Our main results on snippet redundancy are consistent with those reported by Clarke et al. [9], although they worked with the much smaller TREC collection. They examined a subset of the TREC-9 queries requiring a person's name as the answer. They varied the number of passages retrieved (which they call depth) from 25 to 100, and observed some improvements in MRR. When the passages they retrieved were small (250 or 500 bytes) they found improvement, but when the passages were larger (1000 or 2000 bytes) no improvements were observed. The snippets we used are shorter than 250 bytes, so the results are consistent. Clarke et al. [9] also explored a different notion of redundancy (which they refer to as c_i). c_i is the number of different passages that match an answer. Their best performance is achieved when both c_i and term weighting are used to rank passages. We too use the number of snippets that an n-gram occurs in. We do not, however, use global term weights, but have tried other techniques for weighting snippets as described below.

5.2 TREC vs. Web Databases

Another way to explore the importance of redundancy is to run our system directly on the TREC documents. As noted earlier, there are three orders of magnitude more documents on the Web than in the TREC QA collection. Consequently, there will be fewer alternative ways of saying the same thing and fewer matching documents available for mining the candidate n-grams. We suspect that this lack of redundancy will limit the success of our approach when applied directly on TREC documents.

While corpus size is an obvious and important difference between the TREC and Web collections there are other differences as well. For example, text analysis, ranking, and snippet extraction techniques will all vary somewhat in ways that we can not control. To better isolate the size factor, we also ran our system against another Web search engine.

For these experiments we used only the AND rewrites and looked at the first 100 snippets. We had to restrict ourselves to AND rewrites because some of the search engines we used do not support the inclusion of stop words in phrases, e.g., "*created +the character +of Scrooge*".

5.2.1 TREC Database

The TREC QA collection consists of just under 1 million documents. We expect much less redundancy here compared to the Web, and suspect that this will limit the success of our approach. An analysis of the TREC-9 query set (201-700) shows that no queries have 100 judged relevant documents. Only 10 of the 500 questions have 50 or more relevant documents, which the results in Figure 2 suggest are required for the good system performance. And a very large number of queries, 325, have fewer than 10 relevant documents.

We used an Okapi backend retrieval engine for the TREC collection. Since we used only Boolean AND rewrites, we did not take advantage of Okapi's best match ranking algorithm. However, most queries return fewer than 100 documents, so we wind up examining most of the matches anyway.

We developed two snippet extraction techniques to generate query-relevant summaries for use in n-gram mining. A *Contiguous* technique returned the smallest window containing all the query terms along with 10 words of context on either side. Windows that were greater than 500 words were ignored. This approach is similar to passage retrieval techniques albeit without differential term weighting. A *Non-Contiguous* technique returned the union of two word matches along with 10 words of context on either side. Single words not previously covered are included as well. The search engine we used for the initial Web experiments returns both contiguous and non-contiguous snippets. Figure 3 shows the results of this experiment.

	AND Rewrites Only, Top 100		
	MRR	NumCorrect	PropCorrect
Web1	0.450	281	0.562
TREC, Contiguous Snippet	0.186	117	0.234
TREC, Non-Contiguous Snippet	0.187	128	0.256

Figure 3: Web vs. TREC as data source

Our baseline system using all rewrites and retrieving 100 snippets achieves 0.507 MRR (Figure 2). Using only the AND query rewrites results in worse performance for our baseline system with 0.450 MRR (Figure 3). More noticeable than this difference is the drop in performance of our system using TREC as a data source compared to using the much larger Web as a data source. MRR drops from 0.450 to 0.186 for contiguous snippets and 0.187 for non-contiguous snippets, and the proportion of questions answered correctly drops from 56% to 23% for contiguous snippets and 26% for non-contiguous snippets. It is worth noting that the TREC MRR scores would still place this system in the top half of the systems for the TREC-9 50-byte task, even though we tuned our system to work on much larger collections. However, we can do much better simply by using more data. The lack of redundancy in the TREC collection accounts for a large part of this drop off in performance. Clarke et al. [10] have also reported better performance using the Web directly for TREC 2001 questions.

We expect that the performance difference between TREC and the Web would increase further if all the query rewrites were used. This is because there are so few exact phrase matches in TREC relative to the Web, and the precise matches improve performance by 13% (0.507 vs. 0.450).

We believe that database size per se (and the associated redundancy) is the most important difference between the TREC and Web collections. As noted above, however, there are other differences between the systems such as text analysis, ranking, and snippet extraction techniques. While we can not control the text analysis and ranking components of Web search engines, we can use the same snippet extraction techniques. We can also use a different Web search engine to mitigate the effects of a specific text analysis and ranking algorithms.

5.2.2 Another Web Search Engine

For these experiments we used the MSNSearch search engine. At the time of our experiments, the summaries returned were independent of the query. So we retrieved the full text of the top 100 web pages and applied the two snippet extraction techniques described above to generate query-relevant summaries. As before, all runs are completely automatic, starting with queries, retrieving web pages, extracting snippets, and generating a ranked list of 5 candidate answers. The results of these experiments are shown in Figure 4. The original results are referred to as Web1 and the new results as Web2.

	AND Rewrites Only, Top 100		
	MRR	NumCorrect	PropCorrect
Web1	0.450	281	0.562
TREC, Contiguous Snippet	0.186	117	0.234
TREC, Non-Contiguous Snippet	0.187	128	0.256
Web2, Contiguous Snippet	0.355	227	0.454
Web2, Non-Contiguous Snippet	0.383	243	0.486

Figure 4: Web vs. TREC as data source

The Web2 results are somewhat worse than the Web1 results, but this is expected given that we developed our system using the Web1 backend, and did not do any tuning of our snippet extraction algorithms. In addition, we believe that the Web2 collection indexed somewhat less content than Web1 at the time of our experiments, which should decrease performance in and of itself. More importantly, the Web2 results are much better than the TREC results for both snippet extraction techniques, almost doubling MRR in both cases. Thus, we have shown that QA is more successful using another large Web collection compared to the small TREC collection. The consistency of this result across Web collections points to size and redundancy as the key factors.

5.2.3 Combining TREC and Web

Given that the system benefits from having a large text collection from which to search for potential answers, then we would expect that combining both the Web and TREC corpus would result in even greater accuracy. We ran two experiments to test this. Because there was no easy way to merge the two corpora, we instead combined the output of QA system built on each corpus. For these experiments we used the original Web1 system and our TREC system. We used only the AND query rewrites, looked at the Top1000 search results for each rewrite, and used a slightly different snippet extraction technique. For these parameter settings, the base TREC-based system had a MRR of .262, the Web-based system had a MRR of .416.

First, we ran an oracle experiment to assess the potential gain that could be attained by combining the output of the Web-based and TREC-based QA systems. We implemented a "switching oracle", which decides for each question whether to use the output from the Web-based QA system or the TREC-based QA system, based upon which system output had a higher ranking correct answer. The switching oracle had a MRR of .468, a 12.5% improvement over the Web-based system. Note that this oracle does not precisely give us an upper bound, as combining algorithms (such as that described below) could re-order the rankings of outputs.

Next, we implemented a combining algorithm that merged the outputs from the TREC-based and Web-based systems, by having both systems vote on answers, where the vote is the score assigned to a particular answer by the system. For voting, we defined string equivalence such that if a string X is a substring of Y, then a vote for X is also a vote for Y. The combined system achieved a MRR of .433 (a 4.1% improvement over the Web-based system) and answered 283 questions correctly.

5.3 Snippet Weighting

Until now, we have focused on the quantity of information available and less on its quality. Snippet weights are used in ranking n-grams. An n-gram weight is the sum of the weights for all snippets in which that n-gram appears.

Each of our query rewrites has a weight associated with it reflecting how much we prefer answers found with this particular query. The idea behind using a weight is that answers found using a high precision query (e.g., "Abraham Lincoln was born on") are more likely to be correct than those found using a lower precision query (e.g., "Abraham" AND "Lincoln" AND "born"). Our current system has 5 weights.

These rewrite weights are the only source of snippet weighting in our system. We explored how important these weight are and considered several other factors that could be used as additional sources of information for snippet weighting. Although we specify Boolean queries, the retrieval engine can provide a ranking, based on factors like link analyses, proximity of terms,

location of terms in the document, etc. So, different weights can be assigned to matches at different positions in the ranked list. We also looked at the number of matching terms in the best fixed width window, and the widow size of the smallest matching passage as indicators of passage quality.

Rewrite Wts uses our heuristically determined rewrite weights as a measure the quality of a snippet. This is the current system default. *Equal Wts* gives equal weight to all snippets regardless of the rewrite rule that generated them. To the extent that more precise rewrites retrieve better answers, we will see a drop in performance when we make all weights equal. *Rank Wts* uses the rank of the snippet as a measure of its quality, *SnippetWt = (100-rank)/100*. *NMatch Wts* uses the number of matching terms in a fixed-width window as the measure of snippet quality. *Length Wts* uses a measure of the length of the snippet needed to encompass all query terms as the measure of snippet quality. We also look at combinations of these factors. For example, *Rewrite+Rank Wts* uses both rewrite weight and rank according to the following formula, *SnippetWt = RewriteScore + (100-rank)/100*. All of these measures are available from query-relevant summaries returned by the search engine and do not require analyzing the full text of the document. The results of these experiments are presented in Figure 4.

Weighting	MRR	NumCorrect	PropCorrect
Equal Wts	0.489	298	0.596
Rewrite Wts (Default)	0.507	307	0.614
Rank Wts	0.483	292	0.584
Rewrite + Rank Wts	0.508	302	0.604
NMatch Wts	0.506	301	0.602
Length Wts	0.506	302	0.604

Figure 5: Snippet Weighting

Our current default 5-level weighting scheme which reflects the specificity of the query rewrites does quite well. Equal weighting is somewhat worse, as we expected. Interestingly search engine rank is no better for weighting candidate n-grams than equal weighting. None of the other techniques we looked at surpasses the default weights in both MRR and PropCorrect. Our heuristic rewrite weights provide a simple and effective technique for snippet weighting, that can be used with any backend retrieval engine.

Most question answering systems use IR-based measures of passage quality, and do not typically evaluate the best measure of similarity for purposes of extracting answers. Clarke et al. [9] noted above is an exception. Soubbotin and Soubbotin [18] mention different weights for different regular expression matches, but they did not describe the mechanism in detail nor did they evaluate how useful it is. Harabagiu et al. [11] have a kind of backoff strategy for matching which is similar to weighting, but again we do not know of parametric evaluations of its importance in their overall system performance. The question of what kinds of passages can best support answer mining for question answering as opposed to document retrieval is an interesting one that we are pursuing.

6. DISCUSSION AND FUTURE DIRECTIONS

The design of our question answering system was motivated by the goal of exploiting the large amounts of text data that is available on the Web and elsewhere as a useful resource. While many question answering systems take advantage of linguistic resources, fewer depend primarily on data. Vast amounts of data provide several sources of redundancy that our system capitalizes on. Answer redundancy (i.e., multiple, differently phrased, answer occurrences) enables us to use only simple query rewrites for matching, and facilitates the extraction of candidate answers.

We evaluated the importance of redundancy in our system parametrically. First, we explored the relationship between the number of document snippets examined and question answering accuracy. Accuracy improves sharply as the number of snippets included for n-gram analysis increases from 1 to 50, somewhat more slowly after that peaking at 200 snippets, and then falls off somewhat after that. More is better up to a limit. We believe that we can increase this limit by improving our weight accumulation algorithm so that matches from the least precise rewrites do not dominate. Second, in smaller collections (like TREC), the accuracy of our system drops sharply, although it is still quite reasonable in absolute terms. Finally, snippet quality is less important to system performance than snippet quantity. We have a simple 5-level snippet weighting scheme based on the specificity of the query rewrite, and this appears to be sufficient. More complex weighting schemes that we explored were no more useful.

The performance of our system shows promise for approaches to question answering which makes use of very large text databases even with minimal natural language processing. Our system does not need to maintain its own index nor does it require global term weights, so it can work in conjunction with any backend retrieval engine. Finally, since our system does only simple query transformations and n-gram analysis, it is efficient and scalable.

One might think that our system has limited applicability, because it works best with large amounts of data. But, this isn't necessarily so. First, we actually perform reasonably well in the smaller TREC collection, and could perhaps tune system parameters to work even better in that environment. More interestingly, Brill et al. [6] described a projection technique that can be used to combine the wealth of data available on the Web with the reliability of data in smaller sources like TREC or an encyclopedia. The basic idea is to find candidate answers in a large and possibly noisy source, and then expand the query to include likely answers. The expanded queries can then be used on smaller but perhaps more reliable collections – either directly to find support for the answer in the smaller corpus, or indirectly as a new query which is issued and mined as we currently do. This approach appears to be quite promising. Our approach seems least applicable in applications that involve a small amount of proprietary data. In these cases, one might need to do much more sophisticated analyses to map user queries to the exact lexical form that occur in the text collection rather than depend on primarily on redundancy as we have done.

Although we have pushed the data-driven perspective, more sophisticated language analysis might help as well by providing more effective query rewrites or less noisy data for mining.

Most question answering systems contain aspects of both – we use some linguistic knowledge in our small query typology and answer filtering, and more sophisticated systems often use simple pattern matching for things like dates, zip codes, etc.

There are a number of open questions that we hope to explore. In the short term, we would like to look systematically at the contributions of other system components. Brill et al. [5] have started to explore individual components in more detail, with interesting results. In addition, it is likely that we have made several sub-optimal decisions in our initial implementation (e.g., omitting most stop words from answers, simple linear accumulation of scores over matching snippets) that we would like to improve. Most retrieval engines have been developed with the goal of finding topically relevant documents. Finding accurate answers may require somewhat different matching infrastructure. We are beginning to explore how best to generate snippets for use in answer mining. Finally, time is an interesting issue. We noted earlier how the correct answer to some queries changes over time. Time also has interesting implications for using redundancy. For example, it would take a while for a news or Web collection to correctly answer a question like "Who is the U. S. President?" just after an election.

An important goal of our work is to get system designers to treat data as a first class resource that is widely available and exploitable. We have made good initial progress, and there are several interesting issues remaining to explore.

7. REFERENCES

[1] AAAI Spring Symposium Series (2002). *Mining Answers from Text and Knowledge Bases*.

[2] S. Abney, M. Collins and A. Singhal (2000). Answer extraction. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP 2000)*, 296-301.

[3] ACL-EACL (2002). *Workshop on Open-domain Question Answering*.

[4] E. Agichtein, S. Lawrence and L. Gravano (2001). Learning search engine specific query transformations for question answering. In *Proceedings of the 10th World Wide Web Conference (WWW10)*, 169-178.

[5] E. Brill, S. Dumais and M. Banko (2002). An analysis of the AskMSR question-answering system. In *Proceedings of 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*.

[6] E. Brill, J. Lin, M. Banko, S. Dumais and A. Ng (2002). Data-intensive question answering. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[7] S. Buchholz (2002). Using grammatical relations, answer frequencies and the World Wide Web for TREC question answering. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[8] J. Chen, A. R. Diekema, M. D. Taffet, N. McCracken, N. E. Ozgencil, O. Yilmazel, E. D. Liddy (2002). Question answering: CNLP at the TREC-10 question answering track. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[9] C. Clarke, G. Cormack and T. Lyman (2001). Exploiting redundancy in question answering. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'2001)*, 358-365.

[10] C. Clarke, G. Cormack and T. Lynam (2002). Web reinforced question answering. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[11] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus and P. Morarescu (2001). FALCON: Boosting knowledge for question answering. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, 479-488.

[12] E. Hovy, L. Gerber, U. Hermjakob, M. Junk and C. Lin (2001). Question answering in Webclopedia. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, 655-664.

[13] E. Hovy, U. Hermjakob and C. Lin (2002). The use of external knowledge in factoid QA. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[14] C. Kwok, O. Etzioni and D. Weld (2001). Scaling question answering to the Web. In *Proceedings of the 10th World Wide Web Conference (WWW'10)*, 150-161.

[15] M. A. Pasca and S. M. Harabagiu (2001). High performance question/answering. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'2001)*, 366-374.

[16] J. Prager, E. Brown, A. Coden and D. Radev (2000). Question answering by predictive annotation. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'2000)*, 184-191.

[17] D. R. Radev, H. Qi, Z. Zheng, S. Blair-Goldensohn, Z. Zhang, W. Fan and J. Prager (2001). Mining the web for answers to natural language questions. In *Proceeding of the 2001* ACM *CIKM: Tenth International Conference on Information and Knowledge Management*, 143-150

[18] M. M. Soubbotin and S. M. Soubbotin (2002). Patterns and potential answer expressions as clues to the right answers. In *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

[19] E. Voorhees and D. Harman, Eds. (2001). *Proceedings of the Ninth Text REtrieval Conference (TREC-9).* NIST Special Publication 500-249.

[20] E. Voorhees and D. Harman, Eds. (2002). *Proceedings of the Tenth Text REtrieval Conference (TREC 2001).).* NIST Special Publication 500-250.