Question Answering by Passage Selection (MultiText Experiments for TREC-9)

C. L. A. Clarke G. V. Cormack D. I. E. Kisman T. R. Lynam

Computer Science, University of Waterloo, Waterloo, Ontario, Canada
mt@plg.uwaterloo.ca

1 Introduction

MultiText has participated in TREC each year since TREC-4 [3, 2, 7, 8, 6]. For TREC-9 we concentrated our efforts on the question answering (QA) track and also submitted runs for the Web track.

The MultiText system incorporates a unique technique for arbitrary passage retrieval, which was used in all of our TREC-9 experiments. The technique efficiently locates high-scoring passages, where the score of a passage is based on its length and the weights of the terms occurring within it. Passage boundaries are determined by the query, and can start and end at any term position. If a document ranking is required, the score of a document is computed by combining the scores of the passages it contains. Versions of the technique have been described in our previous TREC papers and elsewhere [4]. Our TREC-8 paper [6] provides a concise overview of the current version.

Our TREC-9 QA experiments extended our TREC-8 work. For TREC-8 we simply stripped stopwords from the question, applied the passage retrieval technique, and submitted 250 or 50 bytes centered at each of the top five passages. Considering that no use was made of the structure of the question or the meaning of words within it, relatively good results were achieved. For TREC-9 we applied both question pre-processing, to a generate a query that is more likely to retrieve passages that contain the answer, and passage post-processing, to select the best 250 or 50 byte answer from within a passage. Both the pre-processor and post-processor make use of a question parser, which generates the query to be executed against the target collection and a set of selection rules that are used to drive a set of pattern matching routines in the post-processor.

In addition, we participated in all aspects of the Web Track, submitting runs over both the 10GB and 100GB collections, 10GB runs incorporating link information, and 10GB runs using both title-only and title-description queries.

2 Question Answering Track

Our question answering system consists of three main components (Figure 1). The parsing component performs the pre-processing of questions, feeding the resulting queries and selection rules to the passage retrieval component and the passage selection component. The passage retrieval component executes the queries over the target corpus, retrieving the ten best passages. The passage selection component applies the selection rules to the passages to produce a ranked set of five 50- or 250-byte answers.

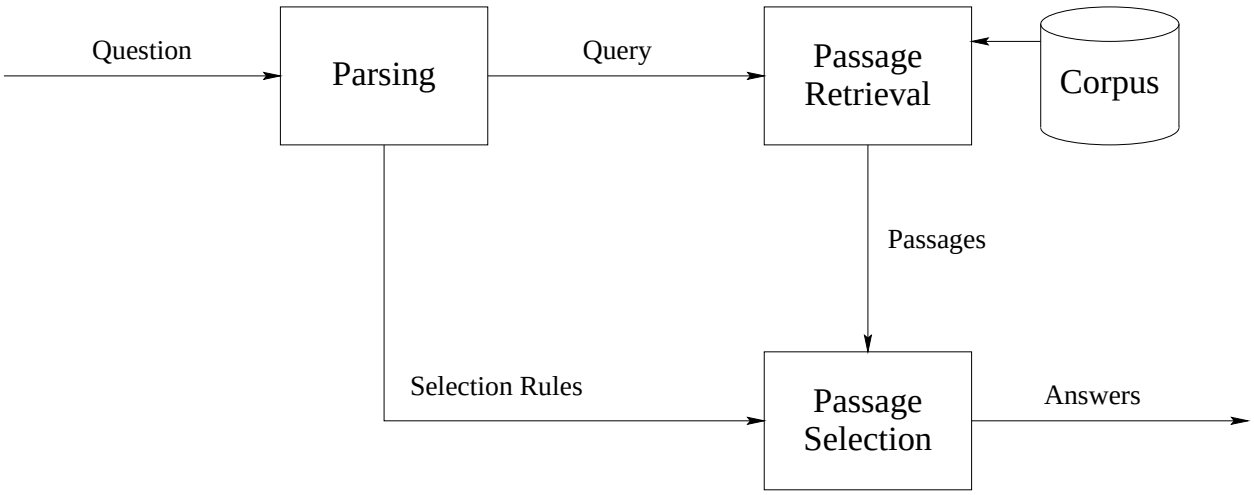

Figure 1: Overview of QA processing

2.1 Passage Retrieval for Question Answering

The MultiText passage retrieval algorithm is used to identify locations in documents where answers are likely to occur. For each question, ten passages are retrieved and passed to the passage selection component for analysis. These ten passages consist of the highest-scoring passages where no two passages are taken from the same document.

For passage retrieval purposes, each document D in the corpus is treated as an ordered sequence of words:

$$D = d_1\ d_2\ d_3 ... d_m.$$

At each word position $(1...m)$ various terms are indexed indicating information about the word. These indexed terms include the word itself, the stemmed word and in some cases a token indicating that the word's position corresponds to the start of a name, a number, a monetary value, or other potential answer value.

A query is treated as a set of terms:

$$Q = \{q_1,\ q_2,\ q_3, ...\},$$

where each term is a word, a phrase, a truncated word, a word with stemming applied or a disjunction of terms. Document and query terms are matched using the expected semantics. For example, the query used for question 425 ("How many months does a normal human pregnancy last?") is

```
months normal human pregnancy "$last" <duration>
```

The term `"$last"` indicates that "last" should be matched under stemming (matching "lasts", "lasting", etc.). The term `<duration>` matches positions in the document where possible time values have been identified during indexing of the corpus.

An *extent* (u, v), with $1 \leq u \leq v \leq m$ is used to represent a subsequence of D beginning at position u and ending at position v:

$$d_u\ d_{u+1}\ d_{u+2} ... d_v.$$

An extent (u, v) *satisfies* a term set $T \subseteq Q$ if the subsequence of D defined by the extent matches all the terms from T. An extent (u, v) is a *cover* for T if (u, v) satisfies T and the subsequence corresponding to (u, v) contains no subsequence that also satisfies T. That is, there does not exist an extent (u', v') with either $u < u' \leq v' \leq v$ or $u \leq u' \leq v' < v$ that satisfies T. The MultiText System uses a fast algorithm to compute covers for all subsets of Q over all documents in a collection [4].

Passages are assigned scores based on their lengths and on the weights assigned to the query terms they match. A term t is assigned an IDF-like weight:

$$w_t \ = \ \log(N/f_t),$$

where f_t is the number of times that t is matched in the corpus and N is the sum of the lengths of all the documents in the corpus. A standard IDF weight is not used since in-document frequencies are not stored in the MultiText index.

The weight assigned to a set of terms $T \subseteq Q$ is the sum of the weights assigned to each term in T:

$$W(T) \ = \ \sum_{t \in T} w_t.$$

If an extent (u, v) is a cover for the term set T then it can be assigned a score combining the length of the extent and the weight of its matching terms:

$$C(T, u, v) = W(T) \ - \ |T| \log(v - u + 1). \tag{1}$$

Once the ten highest-scoring extents from distinct documents were determined, the centerpoint of each extent was computed $(u+v)/2$ and a 200-word passage centered at this point was retrieved from the corpus. These 200-word passages were then used for final passage selection.

The ability of the algorithm to locate potential answers is illustrated by Figure 2. The figure is based on the top passage retrieved for each of the 200 TREC-8 questions. The queries used to retrieve these passages were generated using our TREC-9 parser, described in Section 2.2. A 50-byte window was slid across each passage a byte at a time, and for each location of the window a regular expression was used to check for an answer to the corresponding question. These regular expressions were taken from the QA evaluation script distributed to track participants by Ellen Voorhees. For each possible location of the window's center relative to the center of the retrieved passage, the figure plots the number of questions for which an answer was found in the window. The graph has an obvious spike within ± 100 bytes of the centerpoint.

2.2 Parsing Questions

The parser has two functions: 1) to generate better queries so that the passage retrieval engine can generate the best candidate passages, and 2) to generate selection rules so that the post-processor can select the best 50-byte or 250-byte answers from the passages.

As a baseline for comparison, we use our simple but effective TREC-8 technique. For this baseline, query terms are exactly those in the question, except for stop words, with no stemming or expansion of any sort. The post-processor merely truncates or expands the top five retrieved passages to form the run. After TREC-8 we observed a number of shortcomings in this baseline technique, which we aim to address with the parser.

In the baseline, stopwords like "how", "when", "first" are eliminated, sometimes eliminating the crux of the question at the same time. In the majority of cases the eliminated words are useless

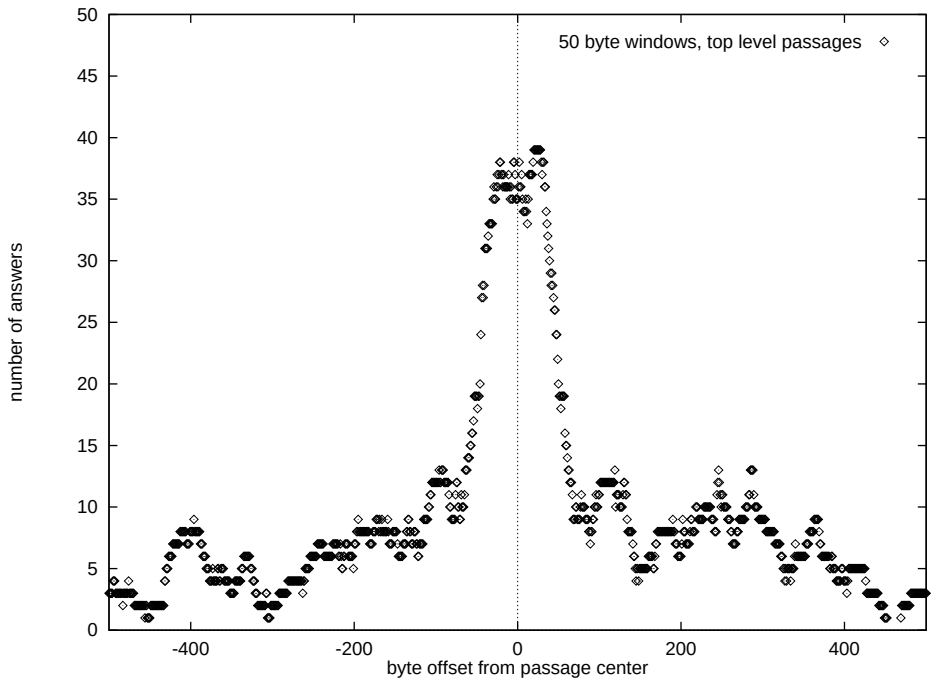

Figure 2: Answer location relative to passage center (TREC-8 data)

as search terms, but in a significant minority of cases the eliminated words would have been useful, and in many other cases the eliminated words contain potentially valuable information about the nature of the question.

In the baseline, words are not stemmed at all. Our experience with the Adhoc, Web, and VLC tasks in previous TREC experiments convinced us that stemming compromises early precision, which is all-important for question answering. Nonetheless, a variant of an exact question word would have often been a better search term. For example, given the question "When did John Kennedy die?", the variant "died" is more likely to appear in the answer than "die". Synonyms like "killed", "shot", or "assassinated" may also glean correct answers. However, in preliminary experiments we found that naive thesaurus expansion, like stemming, compromised early precision, and was overall a detriment to question answering.

We and others [9] have observed that many of the questions can be categorized (Who, What, Where, How far, How many, etc) and that answers to questions of a particular category often contained particular terms. For example, answers to "How far" questions almost certainly contain a number and a unit of distance. Answers to "How many" questions also contain a number and a unit, but the unit term depends on the question rather than the question category alone. For example, the answer to "How many angels can dance on the head of a pin?" almost certainly contains a number followed by "angel" or "angels".

Some categories contain more subtle information about the form of the answer. The answer to "What is the capital of Uraguay?" almost certainly contains the name of a city, or, to be more specific, a capital city. This particular question is common enough in the TREC-8 questions that one might consider handling "capital" as a special case, but we wished to find a more general solution. "What river flows through Kansas City?" is more subtle still. With appropriate analysis one could deduce that the answer is a river, and enumerate the possible answers, or one could make

use of the verb "flows" to deduce that a likely form of the answer is the name of the river followed by some conjugation of "flow".

The parser attempts to infer some of this information from the question and use it to generate the queries and selection rules. The queries and the selection rules capture the results of the parsing in different forms. The queries consist of terms that are likely to appear close to answer. The selection rules explicitly contain category and part-of-speech information, as well as lexical patterns that cannot be matched by the passage retrieval system.

2.2.1 Implementation of the Parser

After considering a number of approaches to question analysis, we developed our own context-free grammar and parser. We considered using shallow (finite-state) patterns but rejected them because: 1) it appeared that the patterns would be as complex as a context-free grammar; 2) we were concerned about over-fitting our analysis to the training data; and 3) we wanted the structure of a parse tree as a framework for question analysis. We also investigated existing natural language parsers, in particular Cooper's CPSG [5] and the Link Grammar Parser [10]. Neither appeared to parse the training questions well, and in both cases we estimated that building an interface for the parser and its output was as complex as parsing the question directly.

Part-of-speech analysis is a necessary complement to context-free parsing, as it is not possible to write a grammar whose vocabulary incorporates all possible words in the question. Instead, generic words like `<verb>`, `<noun>`, etc. are recognized by a separate component. We did not find a publicly available part-of-speech tagger suitable to our needs. Instead we used WordNet to determine whether each word of the question was likely a noun, verb, adjective, or adverb. Parts of speech not recognized by WordNet (articles, pronouns, prepositions) were explicitly enumerated in the grammar. We used a simple strategy suggested in the WordNet documentation [1]: We assumed that the probability of a word being a particular part of speech was proportional to the number of senses listed for that part-of-speech. For example, the word "head" has 30 noun senses, 6 verb senses, 2 adjective senses, and no adverb senses, so we assign probabilities 0.79, .16, .05, and ϵ to these senses. In addition, explicit words in the grammar (like articles, pronouns, and important other words) are given their own categories with high probabilities.

The parser uses Earley's algorithm to determine all possible parses and selects the one that is most probable; the probability of a rule is the product of the probabilities of the terms on its right side, multiplied by an optional weight factor associated with the rule. For example, the probability of matching the rule

```
NOUNPHRASE -> <adjective> NOUN 0.9
```

is the probability of matching `<adjective>` times the probability of matching `NOUN` times 0.9. The grammar contains only 80 production rules (see appendix).

2.2.2 Analysis of the Parse Tree

The most probable derivation tree is walked by an attribute evaluator. The evaluator accumulates each of the important words and its part of speech (noun, verb, adjective, or adverb) and attempts to identify the subject and predicate of the question. Quoted phrases and capitalized sequences of words are identified. The category of the question is identified for various phrasings. For example "Name the capital of Uruguay", "What is the capital of Uruguay?", and "What is Uruguay's capital?" all yield the same result. Special forms involving "How" are recognized as specific categories. Finally, an attribute "instanceof" is computed which identifies a hypernym of the answer. For the

examples above, this attribute is "capital". The parse tree and the computed attributes are written to an intermediate file which is processed by the query generator.

The query generator uses the attributes to formulate queries and selection rules. It deduces more specific answer categories using the "instanceof" and other attributes as well as its knowledge of a few specific words. For example, if the "instanceof" attribute is one of "value", "debt", "earnings", etc. the answer category is assumed to be "money". If the question category is "How long" the answer category may be "distance" or "time interval". For each possible answer category, the generator may add compound terms that retrieve likely answers. For example, the answer category "time interval" causes a term like `("hours"+"minutes"+"weeks"+ ... +"aeons")` to be added to the query. The generator further generates a selection rule that would identify a number followed by one of these terms.

Quoted or capitalized phrases were added as query terms in addition to each of their constituent words. Possessive adjectives were expanded. For example, queries for the capital of Uruguay would all include the term `("Uruguay's"+"of Uruguay"+"of the Uruguay")`. Verbs were stemmed and irregular verbs were expanded to include all conjugations. Nouns, adjectives, and adverbs were not stemmed. Pronouns, articles and prepositions were omitted from the query. We had intended to use WordNet and a number of special-purpose dictionaries to expand a large number of "instanceof" attributes. Unfortunately, time did not permit.

All of the attribute and category information was written to yet another intermediate file, which was read by the post-processor. A separate pattern file translated some answer categories and "instanceof" attributes into lexical patterns.

2.3 Passage Selection

For each question, the passage selection post-processor receives a list of ten ranked passages from the retrieval engine and category information from the parser. In addition, the post-processor consults external databases containing lists of countries, states, cities, proper names, etc. Post-processing then proceeds with the following steps:

1) Determine the answer category from the parser output.
2) Scan the passages for patterns matching the answer category.
3) Assign each possible answer term an initial score based on its rarity.
4) Decrease or increase the the term scores depending on various quality heuristics.
5) Select from the passages the (50-byte or 250-byte) answer that maximizes the sum of the term scores it contains.
6) Set the scores of all terms appearing in the selected answer to zero.
7) Repeat steps 5 and 6 until five answers are selected.

In the first step, the answer categories yielded by the parser are reduced by the pre-processor to one of "Proper" (person, name, company, etc.), "Place" (city, country, date, etc.), "Time" (date, duration, weekday, etc.), "How" (much, many, far, long, etc.), or "Other". The "How" category includes special subcategories for monetary values, numbers, distances and other measurements.

Next the passages are scanned using the patterns for the given category (step 2). These pattern generally consist of regular expressions with simple hand-coded extensions. For example, the pattern for "Proper" is simply `[^A-Za-z0-9][A-Z][A-Za-z]+[^A-Za-z0-9]`, which matches a capital letter followed by one or more letters surrounded by white space or punctuation. Repeated strings are considered more likely to be the answer, as are strings found in highly ranked passages and strings found near the center of passages.

The third step is to assign a score to the term, using the formula

$$c_t \ \log(N/f_t),$$

where f_t is the number of times the term appears in the corpus, N is the sum of the lengths of all the documents in the corpus, and c_t is the number of retrieved passages in which the term appears.

The fourth step modifies the scores using a number of heuristics. First, the score is modified according to its distance from the center of the the retrieved passage. The score is multiplied by $1 - \textit{distance}/1000$, lowering the scores for terms that are farther from the center. Then the score is modified depending on the rank of the passage in which it was found. It is multiplied by $1 - \textit{rank}/1000$, giving a lower score to terms from passages ranked lower by the retrieval engine. The term score is also modified using additional patterns specific to the answer category. Some patterns are "boosters" which increase the score, while some are "reducers" which decrease the score. For example for a "Proper" answer category, the occurrence of "Mr." before the term is a booster, whereas being part of a "Date" pattern is a reducer. The exact values of the boosters and reducers were determined and adjusted manually using the training data.

For each 50- or 250-byte substring of the passages, its score is simply the sum of the scores of the terms within it. The best answer is the substring of the required length with the highest score. This answer is selected for inclusion in the run, and the scores of all terms appearing in it are reduced to zero (step 6). The next best answer is then selected, and this process repeats until five answers are generated (step 7). The purpose of reducing the term scores to zero after selection is to eliminate duplication. However, there is a still a risk that part of the answer may appear in an incorrect passage and the reduction in weight might cause the correct answer to be missed.

Consider the question "Who is the leader of India?" (question 215). The highest-scoring terms and their scores (scaled by 1/1000) are "Sikhs"(27), "Vishwanath"(22), "Pratap"(20), "Tamil"(16), "Farooqui"(8), "Wire"(7), "Nadu"(5), "Madras"(3), "Punjab"(3),"Indian"(2), "Velupillai"(1.6), "urges"(1.5), "Hindu"(1.1), "Prabhakaran"(9.8), "Gandhi"(0.7), "Dixit"(0.7), "Participation"(0.7), and "Singh"(0.6). The top five 50-byte passages returned by the postprocessor are:

1) `. Indian Prime Minister Vishwanath Pratap Singh f`
2) `. Front. INDIA LEADER URGES SIKHS' PARTICIPATION.`
3) `PUNJAB PEACE. From Times Staff and Wire Reports.`
4) `unist Party of India) leader, Mr M. Farooqui. "Bu`
5) `d Monday. J.N. Dixit said Velupillai Prabhakaran,`

2.4 Results

The officials results from our QA runs are summarized in Figure 3. Two of the runs (`uwmt9qas0` and `uwmt9qal0`) were generated by the method described in Section 2.3. The other two (`uwmt9qas1` and `uwmt9qal1`) were generated by a similar technique that used a different (and less successful) approach to pattern matching.

Along with the official strict and lenient mean reciprocal ranks, in its last column the figure lists the results of our own internal judging of the 250-byte runs, which was undertaken immediately after the QA runs were submitted. Since we could only submit two official runs, we performed our own judging to examine the individual effects of the various QA processing components. The results of are presented in Figure 4.

The values in Figure 4 are for questions 201-700 only, since the remaining questions are rephrasing of previous questions. The first row of the figure gives the mean reciprocal rank for a baseline

run id	length (bytes)	strict official judgements	lenient official judgements	MultiText judgements
uwmt9qas0	50	0.321	0.339	
uwmt9qas1	50	0.257	0.264	
uwmt9qal0	250	0.456	0.475	0.486
uwmt9qal1	250	0.460	0.465	0.456

Figure 3: Mean reciprocal ranks over all QA questions

	strict official judgements	lenient official judgements	MultiText judgements
baseline			0.414
parser-generated queries			0.464 (+12%)
uwmt9qal0	0.464	0.471	0.502 (+21%)
uwmt9qal1	0.457	0.460	0.472 (+14%)

Figure 4: Mean reciprocal ranks for 250 byte runs over QA questions 201-700 only

run that duplicates the method used for our TREC-8 QA submissions. The queries used for this run were obtained by stripping the stopwords from the questions, and answer selection consisted of truncating or extending the top five retrieved passage to the appropriate length. The next row shows the effect of the parser. For this run, the queries were generated by the parser, but passage selection post-processing was not preformed. As with the baseline run, the answer was produced by truncation or extension of the retrieved passages. The last two rows give the mean reciprocal rank over questions 201-700 using both the official judgements and our judgements. For our best 250-byte run, the use of the question parser gave a 12% improvement over the baseline technique, and passage selection gave a further 8% improvement, for an overall improvement of 21%.

3 Web Track

We submitted six runs for the small (10GB) Web track and three runs for the large (100GB) Web track. Document ranking for all of the runs was based on the same version of our *cover density ranking* algorithm that we used for our TREC-8 experiments [6]. New for TREC-9 was the use of a 4-gram index for the small Web track and some limited use of document links. Cover density ranking is an extension of the passage retrieval technique described in section 2.1, in which the score of a document is computed from the scores of the passages it contains.

Figure 5 summaries our submissions for the small Web track. Three runs use only page content for ranking. The first (uwmt9w10g0) uses words from the title only; the second (uwmt9w10g1) uses words from both the title and description. For both runs, query terms were generated by normalizing the words to lower case and eliminating stopwords. The terms were not stemmed since it was our experience in earlier TREC Web experiments that stemming adversely effects the precision in the top 20 or so documents. The third content-only run (uwmt9w10g4) is based on the title only, but in contrast to the other runs, this run uses 4-gram indexing for retrieval.

Web queries and documents often contain significant spelling errors, and our use of 4-grams was planned to address this problem. For indexing, each word in each document was split into

run id	run description	avg. prec.	prec. @ 10
uwmt9w10g0	title-only, content-only	0.165	0.238
uwmt9w10g1	title-description, content-only	0.133	0.260
uwmt9w10g2	title-only, content-link (from uwmt9w10g0)	0.163	0.236
uwmt9w10g3	title-description, content-link (from uwmt9w10g1)	0.134	0.262
uwmt9w10g4	4-gram, title-only, content-only	0.181	0.240
uwmt9w10g5	4-gram, title-only, content-link (from uwmt9w10g4)	0.179	0.240

Figure 5: Small Web track results

overlapping 4-grams. For example, the word "tomato" would be split into the terms "toma", "omat" and "mato". Each word was split individually; 4-grams did not span multiple words or contain punctuation or white space. Cover density ranking requires that the word position of each term be available in the index, and in this case all 4-grams generated by a word were treated as occurring at the word's position. Words consisting of less than 4 characters were indexed directly.

We were surprised at the relative difference between the title-only run using words (uwmt9w10g0) and the title-only run using 4-grams (uwmt9w10g4). The 4-gram run achieved a 9% better average precision but similar precision at 5-20 documents. We had expected the 4-gram run to exhibit better performance on those queries that contained significant spelling errors, and this hypothesis was confirmed in a limited way. However, only a few of the titles actually contained spelling errors, and most of the performance difference appears to be due the matching of morphological variants by the 4-gram queries. This result is in contrast to our earlier experience with stemming.

We were also surprised at the relative difference between the title-description run (uwmt9w10g1) and the corresponding title-only run (uwmt9w10g0), with the title-only run achieving a 24% better average precision.

The remaining three small Web track runs (uwmt9w10g2, uwmt9w10g3, and uwmt9w10g5) represent our first attempt to use link information for TREC experiments. Each run was generated from one of the content-only runs by using link information in an additional re-ranking step. The re-ranking had little impact on retrieval effectiveness.

The table of Figure 6 summaries our submissions for the large Web track. Our three TREC-9 submissions repeat our submissions for the TREC-8 large Web track and the figure directly compares the two years. The runs may also be compared with our small Web word-based, title-only run (uwmt9w10g0) which uses similar queries and essentially the same ranking method. All three large Web runs use the topic words mapped to lower case with stopwords eliminated. The first run (uwmt9w100g0) uses these terms without change. The second run (uwmt9w100g1) extends these terms with a simple stemmer for plurals. The third run (uwmt9w100g2) uses only the three terms having the greatest term weights (w_t). The difference between our TREC-8 and TREC-9 results is quite dramatic considering that the only difference is in the selection of the query subset for judging and the actual judging itself.

	TREC-9		TREC-8	
run id	avg. prec.	prec. @ 10	avg. prec.	prec. @ 10
uwmt9w100g0	0.351	0.454	0.478	0.586
uwmt9w100g1	0.328	0.427	0.487	0.580
uwmt9w100g2	0.336	0.427	0.470	0.590

Figure 6: Large Web track results

References

[1] Richard Beckwith, George Miller, and Randee Tengi. Design and implementation of the WordNet lexical database and searching software. Included in wordNet software distribution. See `http://www.cogsci.princeton.edu/~wn/`.

[2] Charles L. A. Clarke and Gordon V. Cormack. Interactive substring retriveal. In *Fifth Text REtrieval Conference (TREC-5)*, pages 295–304, Gaithersburg, Maryland, November 1996.

[3] Charles L. A. Clarke, Gordon V. Cormack, and Forbes J. Burkowski. Shortest substring ranking. In *Fourth Text REtrieval Conference (TREC-4)*, pages 295–304, Gaithersburg, Maryland, November 1995.

[4] Charles L. A. Clarke, Gordon V. Cormack, and Elizabeth A. Tudhope. Relevance ranking for one to three term queries. In *Fifth RIAO Conference*, pages 388–400, Montreal, June 1997. A version of this paper will appear in *Information Processing and Management*, 2000.

[5] R. Cooper. *Classification-based Phrase Structure Grammar — An Extended Revised Version of HPSG*. PhD thesis, University of Edinburgh, 1990.

[6] G. V. Cormack, C. L. A. Clarke, C. R. Palmer, and D. I. E. Kisman. Fast automatic passage ranking. In *Eighth Text REtrieval Conference (TREC-8)*, Gaithersburg, Maryland, November 1999.

[7] Gordon V. Cormack, Charles L. A. Clarke, Christopher R. Palmer, and Samual S. L. To. Passage-based refinement. In *Sixth Text REtrieval Conference (TREC-6)*, pages 303–319, Gaithersburg, Maryland, November 1997. A version of this paper will appear in *Information Processing and Management*, 2000.

[8] Gordon V. Cormack, Christopher R. Palmer, Michael Van Biesbrouck, and Charles L. A. Clarke. Deriving very short queries for high precision and recall. In *Seventh Text REtrieval Conference (TREC-7)*, pages 121–132, Gaithersburg, Maryland, November 1998.

[9] John Prager, Eric Brown, Anni Coden, and Dragomir Radev. Question-answering by predictive annotation. In *23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 184–191, Athens, July 2000.

[10] Davy Temperley, Daniel Sleator, and John Lafferty. The link grammar parser. See `http://www.link.cs.cmu.edu/link`.

Appendix - The Grammar

```
START -> NUH SENTENCE

SENTENCE -> Capital S .9 | S

S -> VP
   | NP .8
   | name NP
   | what TOBE the NAHE AJPREP NP 1.2
   | NP VP
   | AV DID NP VP
   | NP DID NP VP
   | NP TOBE VV
   | NP TOBE NP
   | AJ TOBE NP
   | AJP TOBE NP

NAHE -> name | names | term | terms | abbreviation | acronym

DID -> did | does | will | were | was | do | has | had | have

TOBE -> is | was | will be | were | are | have | has
      | contains | Punc-' s

VP -> VV
    | VV NP
    | VV NP AVP

VV -> V
    | VV AVP
    | VV CONJ V

NP -> NN
    | NP COMMA AJP COMMA
    | NP CONJ NN

CONJ -> and | or | Punc-,

NN -> N
    | ART N

N -> Noun | PRON | AJ N | NUH | STR | Capital Noun | Capital PRON
   | Gerund Noun | name | Initial Noun | Acronym Noun | Capital a | US

US ->  Acronym us 1.1 |
    | Initial u Initial s 1.1
    | Capital united Capital states 1.1 | Acronym usa 1.1
    | Initial u Initial s Initial a 1.2

AJ -> Adjective
    | Noun | Initial Noun | Capital Noun | Acronym Noun
    | PRON | Initial PRON | Capital PRON | Acronym PRON
    | Noun Punc-' | Initial Noun Punc-' | Capital Noun Punc-'
    | Noun Punc-' s | Initial Noun Punc-' s | Capital Noun Punc-' s
    | Acronym Noun Punc-' | Acronym Noun Punc-' s
    | AV Adjective
    | Capital Adjective
    | Initial Adjective
    | Acronym Adjective
    | US
    | NUH

AJP ->  AJPREP NP
     | AJINTRO VP
     | AJINTRO NN VP
     | AJPREP Gerund VP
     | Gerund VP

AVP -> COMMA AV COMMA
     | AVPREP NP
     | AVINTRO NP VP
     | AVINTRO VP

V -> Verb
   | Verb AV  1.1
   | AV V
   | Verb V
   | to V
   | Gerund Verb
   | Capital Verb 0.2
   | Initial Verb 0.2
   | Acronym Verb 0.2

AV -> Adverb | AV Adverb | Capital Adverb 0.2
    | Initial Adverb 0.2 | Acronym Adverb 0.2

AVINTRO -> while | during | after | before | when | like | as | upon

AJINTRO -> who | whom | which | that | whose | Punc-, | to | at | off

AJPREP -> of | over | by | at | in | between | under
        | from | for | upon | on | with | about
| around | Punc-, | off | up | down | into

AVPREP -> over | by | at | in | to | into | under
        | from | for | via | on | onto | with | about
| around | Punc-, AVPREP | since | during | upon

PRON -> his | her | it | its | their | this
      | that | what | our | us | them | he
      | she | who | whom | hers | which

ART -> the | a | an

NUH -> DIG

STR -> COMMA Punc-" STUFF Punc-" COMMA

DIG -> Number

STUFF -> W | STUFF W

W -> Punc-, | Punc-. | Punc-?
     | ART | AVPREP | AJPREP | AVINTRO | AJINTRO | Punc-, | Capital
     | Acronym | Initial | Noun | Verb | Adverb | Adjective | Number

COMMA ->
      | Punc-,
```