The Problem of Precision in Restricted-Domain Question-Answering. Some Proposed Methods of Improvement

DOAN-NGUYEN Hai and **Leila KOSSEIM**
CLaC Laboratory, Department of Computer Science, Concordia University
Montreal, Quebec, H3G-1M8,
Canada
haidoan@cs.concordia.ca, kosseim@cs.concordia.ca

Abstract

This paper discusses some main difficulties of restricted-domain question-answering systems, in particular the problem of precision performance. We propose methods for improving the precision, which can be classified into two main approaches: improving the Information Retrieval module, and improving its results. We present the application of these methods in a real QA system for a large company, which yielded very good results.

1 Introduction

Restricted-domain Question-Answering (RDQA) works on specific domains and often uses document collections restricted in subject and volume. It has some characteristics that make techniques developed recently for open-domain QA, particularly those within TREC (Text REtrieval Conference, e.g. (TREC, 2002)) competitions, become less helpful. First, in RDQA, correct answers to a question may often be found in only very few documents. Light et al (2001) give evidence that the performance on precision of a system depends greatly on the redundancy of answer occurrences in the document collection[1]. Second, a RDQA system has often to work with domain-specific terminology, including domain-specific word meaning. Lexical and semantic techniques based on general lexicons and thesauri, such as WordNet, may not apply well here. Third, if a QA system is to be used for a real application, e.g. answering questions from clients of a company, it should accept complex questions, of various forms and styles. The system should then return a complete answer, which can be long and complex, because it has to, e.g., clarify the context of the problem posed in the question, explain the options of a service, give instructions, procedures, or suggestions, etc. Contrarily, techniques from TREC competitions, aiming at finding short and precise answers, are often based on the hypothesis that the questions are constituted by a single, and often simple, sentence, and can be categorized into a well-defined and simple semantic classification (e.g. Person, Time, Location, Quantity, etc.).

RDQA has a long history, beginning with systems working over databases (e.g., BASEBALL (Green et al, 1961) and LUNAR (Woods, 1973)). Recently, research in QA has concentrated mostly on open-domain QA, in particular on how to find a very precise and short answer. Nonetheless, RDQA seems to be regaining attention, as shown by this ACL workshop. Researchers are also beginning to recognize the importance of long and complete answers. Lin et al (2003) carried out experiments showing that users prefer an answer within context, e.g., an answer within its containing paragraph. Buchholz and Daelemans (2001) defined some types of complex answers, and proposed that the system presents a list of good candidates to the user, and let him construct the reply by himself. Harabagiu et al (2001) mentioned the class of questions that need a listing answer.

One well-known approach for RDQA was *semantic grammars* (Brown and Burton, 1975), which build pre-defined patterns of questions for a specific task. Simple and easy to implement, this approach can only deal with very small tasks, and a restricted set of questions. The most popular class of techniques for QA – whether it is restricted-domain or open-domain, includes using thesauri and lexicons, classifying documents, and categorizing the questions. Harabagiu et al (2000), for example, use WordNet extensively to generate keyword alternations and infer the expected answer category of a question.

In this paper, we present several methods to improve the precision of a RDQA system which

[1] For example, they estimate that only about 27% of the systems participating in TREC-8 produced a correct answer for questions with exactly one answer occurrence, while about 50% of systems produced a correct answer for questions with 7 answer occurrences. (7 is the average answer occurrences per question in the TREC-8 collection.)

should accept freely complex questions and return complete answers. We use our experiments in developing a real system as demonstration.

2 Overview of the demonstration system

The objective of this system is to reply to clients' questions on services offered by a large company, here Bell Canada. The company provides wide-range services on telephone, wireless, Internet, Web, etc. for personal and enterprise clients. The document collection was derived from HTML and PDF files from the company's website (www.bell.ca). As the structure of these files was so complicated, documents were saved as pure text with no mark-ups, sacrificing some important formatting cues like titles, listings, tables. The collection comprises more than 220 documents, of a total of about 560K characters.

The available question set has 140 questions. It was assured that every question has an answer from the contents of the collection. The form and style of the questions vary freely. Most questions are composed of one sentence, but some are composed of several sentences. The average length of questions is 11.3 words (to compare, that of TREC questions is 7.3 words). The questions ask about what a service is, its details, whether a service exists for a certain need, how to do something with a service, etc. For the project, we divided the question set at random into 80 questions for training and 60 for testing. Below are some examples of questions:

Do I have a customized domain name even with the Occasional Plan of Business Internet Dial?

With the Web Live Voice service, is it possible that a visitor activates a call to our company from our web pages, but then the call is connected over normal phone line?

It seems that the First Rate Plan is only good if most of my calls are in the evenings or weekends. If so, is there another plan for long distance calls anytime during the day?

Although our collection was not very large, it was not so small either so that a strategy of searching the answers directly in the collection could be obvious. Hence we first followed the classic two-step strategy of QA: information retrieval (IR), and then candidate selection and answer extraction. For the first step, we used Okapi, a well-known generic IR engine (www.soi.city.ac.uk/~andym/OKAPI-PACK/, also (Beaulieu et al, 1995)). For each question, Okapi returns an ordered list of answer candidates, together with a relevance score for each candidate and the name of the document containing it. An answer candidate is a paragraph which Okapi considers most relevant to the question.[2]

The candidates were then evaluated by a human judge using a binary scale: correct or incorrect. This kind of judgment is recommended in the context of communications between a company and its clients, because the conditions and technical details of a service should be edited as clearly as possible in the reply to the client. However we did also accept some tolerance in the evaluation. If a question is ambiguous, e.g., it asks about phones but does not specify whether it pertains to wired phones or wireless phones, all correct candidates of either case will be accepted. If a candidate is good but incomplete as a reply, it will be judged correct if it contains the principal theme of the supposed answer, and if missing information can be found in paragraphs around the candidate's text in the containing document.

Table 1 shows Okapi's performance on the training question set. We kept at most the 10 best candidates for each question, because after rank 10 a good answer was very rare. C(n) is the number of candidates at rank n which are judged correct. Q(n) is the number of questions in the training set which have at least one correct answer among the first n ranks. As for answer redundancy, among the 45 questions having at least a correct answer (see Q(10)), there were 33 questions (41.3% of the entire training set) having exactly 1 correct answer, 10 questions (12.5%) having 2, and 2 questions (2.5%) having 3 correct answers. Table 2 gives Okapi's precision on the test question set.

The results show that Okapi's performance on precision was not satisfying, conforming to our discussion about characteristics of RDQA above. The precision was particularly weak for n's from 1 to 5. Unfortunately, these are cases that the system aims at. n=1 means that only one answer will be returned – a totally automatic system. n=2 to 5 correspond to more practical scenarios of a semi-automatic system, where an agent of the company chooses the best one among the n candidates, edits it, and sends it to the client. We stopped at n=5 because a greater number of candidates seems too heavy psychologically to the human agent. Also note that the rank of the candidates is not important here, because they would be equally examined by the agent. This explains why we used Q(n) to measure the precision performance rather than

[2] A paragraph is a block of text separated by double newlines. As formatted files were saved in plain text, original "logical" paragraphs may be joined up into one paragraph, which may affect the precision of the candidates.

other well-known scoring such as mean reciprocal rank (MRR).

Examining the correct candidates, we found that they were generally good enough to be sent to the user as an understandable reply. About 25% of them contained superfluous information for the corresponding question, while 15% were lacking of information. However, only 2/3 of the latter (that is 10% of all) looked difficult to be completed automatically. Building the answer from a good candidate therefore seemed less important than improving the precision of the IR module. We therefore concentrated on how to improve Q(n), n= 1 to 5, of the system.

n	1	2	3	4	5	6	7	8	9	10
C(n)	20	11	5	4	9	3	1	1	4	1
%C(n)	25%	13.8%	6.3%	5%	11.3%	3.8%	1.3%	1.3%	5%	1.3%
Q(n)	20	26	28	32	39	41	42	43	44	45
%Q(n)	25%	32.5%	35%	40%	48.8%	51.3%	52.5%	53.8%	55%	56.3%

Table 1: Precision performance of Okapi on the training question set (80 questions).

n	1	2	3	4	5	6	7	8	9	10
C(n)	18	8	7	2	4	3	3	2	1	1
%C(n)	30%	13.3%	11.7%	3.3%	6.7%	5%	5%	3.3%	1.7%	1.7%
Q(n)	18	23	28	29	32	33	35	36	36	37
%Q(n)	30%	38.3%	46.7%	48.3%	53.3%	55%	58.3%	60%	60%	61.7%

Table 2: Precision performance of Okapi on the test question set (60 questions).

3 Methods for Improving Precision Performance

The first approach to improve the precision performance of the IR module is to use a better engine, e.g. by adjusting the parameters, modifying the formulas of the engine, or replacing a generic engine by a more domain-specific one, etc.

Now suppose that the IR engine is already fixed, e.g. because we have achieved the best engine, or, more practically, because we cannot make changes or afford another engine. The second approach consists in improving the results returned by the IR engine. One main direction is candidate re-ranking, i.e. pushing good candidates in the returned candidate list to the first ranks as much as possible, thus increasing Q(n). To do this, we need some information that can characterize the relevance of a candidate to the corresponding question better than the IR engine did. The most prominent kind of such information may be the domain-specific language used in the working domain of the QA system, particularly its vocabulary, or even more narrowly, its terminological set.

In the following, we will present our development of the second approach on the Bell Canada QA system first, because it seems less costly than the first one. However, we will present some implementations of the first approach later.

4 Improving Precision by Re-ranking Candidates

We experimented with two methods of re-ranking, one with a strongly specific terminological set, and one with a good document characterization.

4.1 Re-ranking using specific vocabulary

In the first experiment, we noted that the names of specific Bell services, such as *'Business Internet Dial'*, *'Web Live Voice'*, etc., could be used as a relevance characterizing information, because they occurred very often in almost every document and question, and a service was often presented or mentioned in only one or a few documents, making these terms very discriminating. To have a generic concept, let's call these names *'special terms'*. Luckily, these special terms occurred normally in capital letters, and could be automatically extracted easily. After a manual filtering, we obtained more than 450 special terms.

We designed a new scoring system which raises the score of the candidates containing occurrences of special terms found in the corresponding question, as follows:

(1) $\mathbf{Score\_of\_candidate[i] = DC \times (OW \times Okapi\_score + RC[i] \times Term\_score + 1)}$

Thus, the score of candidate i in the ranked list returned by Okapi depends on: (i) The original **Okapi_score** given by Okapi, weighted by some integer value **OW**. (ii) A **Term_score** that measures the importance of common occurrences of special terms, and, with less emphasis, other noun phrases and open-class words, in the question and the candidate. It is weighted by some integer value **RC[i]** (for *rank coefficient*) that represents the role of the relative ranking of Okapi. (iii) A *document coefficient* **DC** that indicates the relative importance of a candidate i coming or not coming from a document which contains at least a special term occurring in the question. DC is thus represented by a 2-value pair; e.g., the pair (1, 0) corresponds to the extreme case of keeping only candidates coming from a document which contains at least one special term in the question, and throwing out all others. We ran the system with 20 different values of DC, 50 of RC, and OW from 0 to 60, on the training question set. See (Doan-Nguyen and Kosseim, 2004) for a detailed explanation of how formula (1) was derived, and how to design the values of DC, RC, and OW.

Formula (1) gave very good improvements on the training set (Table 3), but just modest results when running the system with optimal training parameters on the test set (Table 4). Note: $\Delta Q(n)$ = System's $Q(n)$ – Okapi's $Q(n)$; $\%\Delta Q(n) = \Delta Q(n)$/Okapi's $Q(n)$.[3]

n	1	2	3	4	5
Q(n)	30	40	42	43	44
ΔQ(n)	10	14	14	11	5
%ΔQ(n)	50%	53.8%	50%	34.4%	12.8%

Table 3: Best results of formula (1) on the training set.

n	1	2	3	4	5
Q(n)	22	29	32	33	34
ΔQ(n)	4	6	4	4	2
%ΔQ(n)	22.2%	26.1%	14.3%	13.8%	6.3%

Table 4: Results of formula (1) on the test set.

[3] Okapi allows one to give it a list of phrases as indices, in addition to indices automatically created from single words. In fact, the results in Tables 1 and 2 correspond to this kind of indexing, in which we provided Okapi with the list of special terms. These results are much better than those of standard indexing, i.e. without the special term list.

4.2 Re-ranking with a better document characterization

In formula (1), the coefficient DC represents an estimate of the relevance of a document to a question based only on special terms; it cannot help when the question and document do not contain special terms. To find another document characterization which can complement this, we tried to map the documents into a system of concepts. Each document says things about a set of concepts, and a concept is discussed in a set of documents. Building such a concept system seems feasible within closed-domain applications, because the domain of the document collection is pre-defined, the number of documents is in a controlled range, and the documents are often already classified topically, e.g. by their creator. If no such classification existed, one can use techniques of building hierarchies of clusters (e.g. those summarized in (Kowalski, 1997)).

We used the original document classification of Bell Canada, represented in the web page URLs, as the basis for constructing the concept hierarchy and the mapping between it and the document collection. Below is a small excerpt from the hierarchy:

```
BellAll
 Personal
  Personal-Phone
   Personal-Phone-LongDistance
    Personal-Phone-LongDistance-BasicRate
    Personal-Phone-LongDistance-FirstRate
```

In general, a leaf node concept corresponds to one or very few documents talking about it. A parent concept corresponds to the union of documents of its child concepts. Note that although many concepts coincide in fact with a special term, e.g. *'First Rate'*, many others are not special terms, e.g. *'phone'*, *'wireless'*, *'long distance'*, etc.

The use of the concept hierarchy in the QA system was based on the following assumption: *A question can be well understood only when we can recognize the concepts implicit in it.* For example, the concepts in the question:

> *It seems that the First Rate Plan is only good if most of my calls are in the evenings or weekends. If so, is there another plan for long distance calls anytime during the day?*

include `Personal-Phone-LongDistance` and `Personal-Phone-LongDistance-FirstRate`. Once the concepts are recognized, it is easy to determine a small set of documents relevant to these concepts, and carry out the search of answers in this set.

To map a question to the concept hierarchy, we postulated that *the question should contain words* *expressing the concepts*. These words may be those constituting the concepts, e.g., *'long'*, *'distance'*, *'first'*, *'rate'*, etc., or synonyms/near synonyms of them, e.g., *'telephone'* to *'phone'*; *'mobile'*, *'cellphone'* to *'wireless'*. For every concept, we built a bag of words which make up the concept, e.g., the bag of words for `Personal-Phone-LongDistance-FirstRate` is *{'personal', 'phone', 'long', 'distance', 'first', 'rate'}*. We also built manually a small lexicon of (near) synonyms as mentioned above.

Now, a question will be analyzed into separate words (stop words removed), and we look for concepts whose bags of words have elements in common with them. (Here we used the Porter stemmed form of words in comparison, and also counted cases of synonyms/near synonyms.) A concept is judged more relevant to a question if: (i) its bag of words has more elements in common with the question's set of words; (ii) the quotient of the size of the common subset mentioned in (i) over the size of the entire bag of words is larger; and (iii) the question contains more occurrences of words in that subset.

From the relevant concept set, it is straightforward to derive the relevant document set for a given question. The documents will be ranked according to the order of the deriving concepts. (If a document is derived from several concepts, the highest rank will be used.) As for the coverage of the mapping, there were only 4 questions in the training set and 6 in the test set (7% of the entire question set) having an empty relevant document set. In fact, these questions seemed to need a context to be understood, e.g., a question like *'What does Dot org mean?'* should be posed in a conversation about Internet services.

Now the score of a candidate is calculated by:

(2) Score_of_candidate[i] = (CC + DC) × (OW × Okapi_score + RC[i] × Term_score + 1)

The value of **CC** *(concept-related coefficient)* depends on the document that provides the candidate. **CC** should be high if the rank of the document is high, e.g. **CC**=1 if rank=1, **CC**=0.9 if rank=2, **CC**=0.8 if rank=3, etc. If the document does not occur in the concept-derived list, its **CC** should be very small, e.g. 0. The sum **(CC + DC)** represents a combination of the two kinds of document characterization. We ran the system with 15 different values of the **CC** vector, with **CC** for rank 1 varying from 0 to 7, and **CC** for other ranks decreasing accordingly. Values for other coefficients are the same as in the previous experiment using formula (1). Results (Tables 5 and 6) are uniformly better than those of formula (1). Good improvements show that the approach is appropriate and effective.

n	1	2	3	4	5
Q(n)	32	41	44	44	44
ΔQ(n)	12	15	16	12	5
%ΔQ(n)	60%	57.7%	57.1%	37.5%	12.8%

Table 5: Best results of formula (2) on the training set.

n	1	2	3	4	5
Q(n)	30	32	35	35	36
ΔQ(n)	12	9	7	6	4
%ΔQ(n)	66.6%	39.1%	25%	20.7%	12.5%

Table 6: Results of formula (2) on the test set.

5 Two-Level Candidate Searching

As the mapping in the previous section seems to be able to point out the documents relevant to a given question with a high precision, we tried to see how to combine it with the IR engine Okapi. In the previous experiments, the entire document collection was indexed by Okapi. Now indexing will be carried out separately for each question: only the document subset returned by the mapping, which usually contains no more than 20 documents, is indexed, and Okapi will search for candidate answers for the question only in this subset. We hoped that Okapi could achieve higher precision in working with a much smaller document set. This strategy can be considered as a kind of two-level candidate searching.

n	1	2	3	4	5
MO Q(n)	18	33	38	45	46
Q(n)	31	42	48	48	48
ΔQ(n)	11	16	20	16	9
%ΔQ(n)	55%	61.5%	71.4%	50%	23.1%

Table 7: Best results of two-level search combined with re-ranking on the training set.

n	1	2	3	4	5
MO Q(n)	20	25	26	29	31
Q(n)	24	28	32	32	33
ΔQ(n)	6	5	4	3	1
%ΔQ(n)	33.3%	21.8%	14.3%	10.3%	3.1%

Table 8: Results of two-level search combined with re-ranking on the test set.

Results show that Okapi did not do better in this case than when it worked with the entire document collection (compare MO Q(n) in Tables 7 and 8 with Q(n) in Tables 1 and 2. MO means *'mapping-then-Okapi'*). We then applied formula (2) to rearrange the candidate list as in the previous section. Although results on the training set (Table 7) are generally better than those of the previous section, results on the test set (Table 8) are worse, which leads to an unfavorable conclusion for this method. (Note that ΔQ(n) and %ΔQ(n) are always comparisons of the new Q(n) with the original Okapi Q(n) in Tables 1 and 2.)

6 Re-implementing the IR engine

The precision of the question-document mapping was good, but the performance of the two-level system based on Okapi in the previous section was not very persuasive. This led us back to the first approach mentioned in Section 3, i.e. replacing Okapi by another IR engine. We would not look for another generic engine because it was not interesting theoretically, but would instead implement a two-level engine using the question-document mapping. As already known, the mapping returns just a small set of relevant documents for a given question; the new engine will search for candidate answers in this set. If the document set is empty, the system takes the candidates proposed by Okapi as results ("Okapi as Last Resort").

We implemented just a simple IR engine. First the question is analyzed into separate words (stop words removed). For every document in the set returned by the question-document mapping, the system scores each paragraph by counting in this paragraph the number of occurrences of words which also appear in the question (using the stemmed form of words). Here 'paragraph' means a block of text separated by one newline, not two as in Okapi sense. Note that texts in the Bell Canada collection contain a lot of short and empty paragraphs. The candidate passage is extracted by taking the five consecutive paragraphs which have the highest score sum. However, if the document is "small", i.e. contains less than 2000 characters, the entire document is taken as the candidate and its score is the sum of scores of all paragraphs.

This choice seemed unfair to previous experiments because about 60% of the collection are such small documents. However, we decided to have a more realistic notion of answer candidates which reflects the nature of the collection and of our current task: in fact, those small documents are often dedicated to a very specific topic, and it seems necessary to present its contents in its entirety to any related question for reasons of understandability, or because of important additional information in the document. Also, a size of 2000 characters (which are normally 70% of a page) seems acceptable for a human judgement in the scenario of semi-automatic systems.[4]

Let's call the score calculated as above **Occurrence_score**. We also considered the role of the rank of the document in the list returned by the question-document mapping. The final score formula is as follows:

(3) Score_of_candidate = RC × (21 - Document_Rank) + Occurrence_score

The portion (21 - Document_Rank) guarantees that high-rank documents contribute high scores. That portion is always positive because we retained no more than 20 documents for every question. **RC** is a coefficient representing the importance of the document rank. Due to time limit – judgement of candidates has to be done manually and is very time consuming, we carried out the experiment with only RC=0, 1, 1.5, and 2, and achieved the best results with RC=1.5.

Results (Tables 9 and 10) show that except the case of n=1 in the test set, the new system performs well in precision. This might be explained partly because it tolerates larger candidates than previous experiments. However what is interesting here is that the engine is very simple but efficient because it does searching on a well selected and very small document subset.

n	1	2	3	4	5
Q(n)	42	55	60	60	61
ΔQ(n)	22	29	32	28	22
%ΔQ(n)	110%	112%	114%	88%	56%

Table 9: Best results of the specific engine on the training set.

n	1	2	3	4	5
Q(n)	23	37	41	42	42
ΔQ(n)	5	14	13	13	10
%ΔQ(n)	27.8%	60.9%	46.4%	44.8%	31.3%

Table 10: Results of the specific engine on the test set.

[4] In fact, candidates returned by Okapi are not uniform in length. Some are very short (e.g. one line), some are very long (more than 2000 characters).

7 Second Approach Revisited: Extending Answer Candidates

The previous experiment has shown that extending the size of answer candidates can greatly ease the task. This can be considered as another method belonging to the second approach – that of improving precision performance by improving the results returned by the IR engine. To be fair, it may be necessary to see how precision performance will be improved if this extending is used in other experiments. We did two small experiments. In the first one, any candidates returned by Okapi (cf. Tables 1 and 2) which came from a document of less than 2000 characters were extended into the entire document. Table 11 shows that improvements are not as good as those obtained by other methods.

n	1	2	3	4	5
Q(n) - A	24	32	35	39	47
ΔQ(n)	4	6	7	7	8
%ΔQ(n)	20%	23.1%	25%	21.9%	20.5%
Q(n) - B	20	27	32	34	37
ΔQ(n)	2	4	4	5	5
%ΔQ(n)	11.1%	17.4%	14.3%	17.2%	15.6%

Table 11: Results of extending Okapi candidates on the training set (A) and test set (B).

In the second experiment, we similarly extended candidates returned by the two-level search process "mapping-then-Okapi" in Section 5. Improvements (Table 12) seem comparable to those of the experiment in Section 5 (Tables 7 and 8), but less good than those of experiments in Sections 4.2 and 6. The two experiments of this section suggest that extending candidates helps improve the precision, but not so much unless it is combined with other methods. We have not yet, however, carried out experiments of combining candidate extending with re-ranking.

n	1	2	3	4	5
Q(n) - A	25	43	48	57	60
ΔQ(n)	5	17	20	25	21
%ΔQ(n)	25%	65.4%	71.4%	78.1%	53.8%
Q(n) - B	24	31	32	38	41
ΔQ(n)	6	8	4	9	9
%ΔQ(n)	25%	34.8%	14.3%	31%	23.1%

Table 12: Results of extending two-level search candidates on the training set (A) and test set (B).

8 Discussions and Conclusions

RDQA, working on small document collections and restricted subjects, seems to be a task no less difficult than open-domain QA. Due to candidate scarcity, the precision performance of a RDQA system, and in particular that of its IR module, becomes a problematic issue. It affects seriously the entire success of the system, because if most of the retrieved candidates are incorrect, it is meaningless to apply further techniques of QA to refine the answers.

In this paper, we have discussed several methods to improve the precision performance of the IR module. They include the use of domain-specific terminology to rearrange the candidate list and to better characterize the question-document relevance relationship. Once this relationship has been well established, one can expect to obtain a small set of (almost) all relevant documents for a given question, and use this to guide the IR engine in a two-level search strategy.

Also, long and complex answers may be a common characteristic of RDQA systems. Being aware of this, one can design appropriate systems which are more tolerant on answer size to achieve a higher precision, and to avoid the need of expanding a short but insufficient answer into a complete one. However, what a good answer should be is still an open question, which would need a lot more study to clarify.

We have also presented applications of these methods in the real QA system for Bell Canada. Good improvements achieved compared to results of the original IR module show that these methods are applicable and effective.

Many other problems on the precision performance of a RDQA system have not been tackled in this paper. Some of them relate to the free form of the questions: how to identify the category of the question (e.g. the mapping 'Who' – Person, 'When' – Time, 'How many' – Quantity, etc.), how to analyze the question into pragmatic parts (pre-suppositions, problem context, question focus), etc. Certainly, they are also problems of open-domain QA if one wants to go further than pre-defined question pattern tasks.

9 Acknowledgements

This project was funded by Bell University Laboratories (BUL) and the Canada Natural Science and Engineering Research Council (NSERC).

References

Beaulieu M., M. Gatford, X. Huang, S.E. Robertson, S. Walker, P. Williams (1995). Okapi at TREC-3. In: *Overview of the Third Text REtrieval Conference (TREC-3)*. Edited by D.K. Harman. Gaithersburg, MD: NIST, April 1995.

Brown, J., Burton, R. (1975). Multiple representations of knowledge for tutorial reasoning. In Bobrow and Collins (Eds), *Representation and Understanding*. Academic Press, New York.

Buchholz, S., Daelemans, W. (2001). Complex Answers: A Case Study using a WWW Question Answering System. *Natural Language Engineering*, 7(4), 2001.

Doan-Nguyen, H., Kosseim, L. (2004). Improving the Precision of a Closed-Domain Question-Answering System with Semantic Information. *Proceedings of RIAO (Recherche d'Information Assistée par Ordinateur (Computer Assisted Information Retrieval)) 2004*. Avignon, France. pp. 850-859.

Green, W., Chomsky, C., Laugherty, K. (1961). BASEBALL: An automatic question answerer. *Proceedings of the Western Joint Computer Conference*, pp. 219-224.

Harabagiu, S., D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Gîrju, V. Rus, P. Morarescu (2000). FALCON: Boosting Knowledge for Answer Engines. *Proceedings of the Ninth Text REtrieval Conference (TREC 2000)*.

Harabagiu, S., D. Moldovan, M. Pasca, M. Surdeanu, R. Mihalcea, R. Girju, V. Rus, F. Lactusu, P. Morarescu, R. Bunescu (2001). Answering Complex, List and Context Questions with LCC's Question-Answering Server. *Proceedings of the Tenth Text REtrieval Conference (TREC 2001)*.

Kowalski, G. (1997). *Information Retrieval Systems – Theory and Implementation*. Kluwer Academic Publishers, Boston/Dordrecht/London.

Light, M., Mann, G., Riloff, E., Breck, E. (2001). Analyses for Elucidating Current Question Answering Technology. *Natural Language Engineering*, 7(4), 2001.

Lin, J., Quan, D., Sinha, V., Bakshi, K., Huynh, D., Katz, B., Karger, D. (2003). The Role of Context in Question Answering Systems. *Proceedings of the 2003 Conference on Human Factors in Computing Systems (CHI 2003)*, April 2003, Fort Lauderdale, Florida.

TREC (2002). Proceedings of The Eleventh Text Retrieval Conference. NIST Special Publication: SP 500-251. E. M. Voorhees and L. P. Buckland (Eds).

Woods W. A. (1973). Progress in natural language understanding: An application to lunar geology. *AFIPS Conference Proceedings*, Vol. 42, pp. 441-450.