# RIAO 2004

Conference Proceedings

*Actes de la conférence*

University of Avignon
(Vaucluse), France
April 26-28, 2004

## Coupling approaches, coupling media and coupling languages for information retrieval

## Combinaisons des approches, combinaisons des média et combinaisons des langues pour la recherche d'information par le contenu

Organized by

**LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE C.I.D.**

# Improving the Precision of a Closed-Domain Question-Answering System with Semantic Information

**Hai Doan-Nguyen & Leila Kosseim**
CLaC Laboratory, Department of Computer Science, Concordia University
Montreal, Québec, Canada, H3G-1M8
{haidoan, kosseim}@cs.concordia.ca

**Abstract**

This paper presents our experiments in applying semantic information to improve the precision of the information retrieval module in a closed-domain question-answering system. That system aims at replying questions on services offered by a large company, here Bell Canada. Our approach consists in finding a set of special terms and building an ontological concept hierarchy, which can effectively characterize the relevance of a retrieved candidate to its corresponding question. Combining these two kinds of semantic information with the information retrieval module has resulted in very good improvements.

## 1. Introduction

This paper presents our experiments in developing a question-answering (QA) system which aims at replying clients' questions on services offered by a company, here Bell Canada. As it is well-known, the precision of the document retrieval stage is essential to the success of a QA system, because it imposes an upper bound to the precision of the entire system. Our approach is to find kinds of semantic information which can effectively characterize the relevance of a retrieved candidate to its corresponding question.

### 1.1. Closed-domain QA

Our system is an instance of closed-domain QA, which works on a document collection restricted in subject and volume. This kind of QA has some characteristics making it different from open-domain QA, which works over a large document collection, including the WWW. In closed-domain QA, correct answers to a question may often be found in only very few documents; the system does not have a large retrieval set abundant of good candidates for selection. Moreover, if the QA system is to be used for answering questions from a company's clients, 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 or procedures, etc. This makes techniques developed recently for open-domain QA, particularly those within TREC (Text REtrieval Conference) competitions (e.g. TREC, 2002) less helpful. These techniques aiming at finding short and precise answers, are often based on the hypothesis that the questions are constituted by a single constituent, and can be categorized into a well-defined and simple semantic classification (e.g. PERSON, TIME, LOCATION, QUANTITY, etc.).

Closed-domain QA has a long history, beginning with systems working over databases (e.g., BASEBALL (Green et al, 1961) and LUNAR (Wood, 1973)). Recently, research in QA has concentrated mostly on problems of open-domain QA, in particular on how to find a very precise and short answer. Nonetheless, researchers begin 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 & Daelemans (2001) define some types of complex answers, and propose that the system presents a list of good candidates to the user, and let him construct the reply by himself. Harabagiu et al (2001) mention the class of questions that need an answer in form of an enumeration (*listing answer*).

### 1.2. Semantic approaches in QA

Use of semantic information in QA systems, or more generally, in any Natural Language Processing applications, has long been studied and continues to be an important research direction, due to its great promises and challenges. One well-known approach was semantic grammars (Brown & 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. Latent Semantic Indexing (Deerwester et al, 1995), a kind of concept indexing, tries to find underlying or "latent" concepts from interrelationships between words. This technique needs a large volume of data for a machine learning process, hence is not applicable for our project. The most popular class of techniques includes using thesauri, lexicons, etc., classifying documents, and categorizing the questions as mentioned above.

Our approach for the Bell Canada QA system consists in finding a set of special terms which characterizes the "working language" of the current task, and building an ontological concept hierarchy which can help better map a question to relevant documents.

## 2. Corpus and question set

Our working corpus contains documents presenting Bell Canada's wide-range services to personal and enterprise clients: telephones, wireless, Internet, etc. The corpus, derived from the company's website (www.bell.ca), has about 560K characters, and comprises more than 220 text documents with no mark-ups. In general, each document corresponds to a web page, but some contain several pages. Most documents are short, of 1K to 5K characters, some are long, up to 24K. As the corpus is a pure text derivation of formatted documents (HTML and PDF), a lot of important information was missing, like titles, subtitles, listings, tables, etc. In several documents, there is "noisy" information, such as general navigation links of the website.

The available question set has 120 questions. It was assured that every question has an answer from the contents of the corpus. The form and style of the questions vary freely. Most questions are composed of one sentence, but there are other composed of many sentences. There are "Wh-questions", "Yes-No questions", questions in form of an imperative (e.g. "Please tell me how..."), etc. 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. Below are some examples of questions:

- *What is Business Internet Dial?*
- *Do I have a customized domain name even with the Occasional Plan of Business Internet Dial?*
- *How can our company separate business mobile calls from personal calls?*
- *Please tell me how I can make an auto reply message when using Bell IP Messaging Webmail.*
- *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?*

For the project, we divided the question set at random into 80 questions for training and 40 for testing.

## 3. Information retrieval module

Although our corpus is not very large, it is not so small either so that a strategy of searching the answers directly in the corpus could be obvious. Moreover, the missing of formatting information mentioned above would make such an approach more difficult. We therefore design our system following the classic strategy of QA systems, with two steps: (1) information retrieval (IR); and (2) candidate selection and answer extraction.

For the first step, we use Okapi, a well-known generic IR engine (www.soi.city.ac.uk/~andym/OKAPI-PACK/, also Beaulieu et al (1995)). After indexing the corpus, we give the training question set (80 questions) to Okapi. For each question, Okapi returns an ordered list of answer candidates, each of which is composed of one or several consecutive paragraphs of a document. Okapi also gives a relevance score for each candidate and the file name of the document containing it – note that a document contributes at most one candidate to a given question. The scores corresponding to our training question set are found in the range of 0 to 40, and precise to thousandths, e.g. 10.741.

The candidates are 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 gives the statistics of Okapi's performance on the training question set. 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. We kept only 10 best candidates at most for each question, because after rank 10, a good answer is very rare.

| 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.** Performance of Okapi on the training question set (80 questions).

The above estimate shows that Okapi's results are not satisfying. Although the rate of 56.3% of questions having a correct answer among the first 10 candidates is acceptable in current QA systems, the rates for n's from 1 to 5 are weak. Unfortunately, these are cases that the system aims at. n=1 means that only one answer will be returned to the user – this corresponds to 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 stop 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 considered important here, because they would be equally examined by the human agent. This explains why we use Q(n) to measure the precision performance of the system rather than other well-known scoring such as mean reciprocal rank (MRR).

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

## 4. Improving the precision of the system with special terms

Our main idea here is to try to push the correct candidates among the 10 best candidates kept for a question to the first ranks as most as possible. To do this, it is necessary to find some kind of information which could effectively characterize the relevance of a candidate to its corresponding question. We note that the names of specific Bell services could be helpful here, because they occur very often in almost every document and question, and a service is often presented or mentioned in only one or a few documents – making these terms very discriminating. To have a generic concept, we will call these names *'special terms'*. Finally, in the corpus, these special terms occur normally in capital letters, e.g. *'Business Internet Dial'*, *'Web Live Voice'*, etc., and can be automatically extracted easily. After a manual filtering, we obtained more than 450 special terms.[1]

### 4.1. Experiment 1: Determining the role of special terms

To show that special terms can help improve the system's performance, we design a series of experiments. First, we wonder whether they can really indicate the relevance of a candidate to a question. A suggestive example is, for the question *'Please explain about Web Hosting Managed Services'*, candidate 4 is the only one containing the special term *'Web Hosting Managed'*, and at the same time, the only correct candidate.

We thus devise a scoring system based only on special terms and open class words (nouns, verbs, adjectives, adverbs) which occur both in the question and the candidate. Let's call all of them 'terms', and let T be the set of common terms, then:

**(1) Candidate's score =** $\sum_{t \in T}$ **basicTermScore(t) + synergyScore(T)**

Due to space limit, we will only give a brief explanation here. *basicTermScore(t)* depends on the type of the term t: in general, it will be largest if t is a special term, then quite large for normal noun phrases, and small in other cases. However, it also increasingly depends on the size (number of words) of t, and the number of occurrences of t in the candidate, because these are good evidence that the candidate is relevant to the question. *synergyScore(T)* is given as a bonus only if T has more than one term. The more different terms occur in common in the candidate and the question, the more likely that this candidate is good for the question. This function therefore increases with the number of terms in T and the size of these terms.

We will call this scoring system *'term scores'*. The candidates returned by Okapi will now be sorted according to their term scores, not Okapi scores. We have tried several different formulas, with different parameters. The results look better than Okapi, only slightly but it is encouraging. Table 2 gives the best results achieved.

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 24 | 29 | 32 | 34 | 37 |
| Improvement | 4 | 3 | 4 | 2 | -2 |

**Table 2.** Results with term scores.

Analyzing the above results shows that the term scoring system did in fact push many correct candidates to the first ranks, e.g. it moved 12 more correct candidates to rank 1. Nevertheless, it also lowered the rank of many correct candidates, e.g. 8 from rank 1, thus making a total improvement of 4. This bad rearrangement was due to two facts: (1) Candidates returned by Okapi are not uniform in length; many correct candidates are too short (e.g., just one line), while many incorrect ones are too long. This would mislead the term scoring system, because a long candidate would have a higher probability of having many terms in common with the question. (2) This scoring system totally ignores computations performed by Okapi. Okapi may have chosen a good document and extracted a good candidate, but this candidate just does not contain many terms in common with the corresponding system.

[1] Okapi allows one to give it a list of phrases (of more than one word) as indices, in addition to indices automatically created from single words. In fact, the results in Table 1 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 (without special term list).

## 4.2. Experiment 2: Combining term scores and Okapi scores

In the hope that a combination of term scores and Okapi scores would yield a better performance, we use the following formula in the second experiment:

**(2) Candidate's score = Term score + OW * Okapi score**

where *OW (Okapi weight)* is an integer running from 0, 1, 2,... to *MaxOW*. We chose MaxOW=60, because at this order the portion (OW * Okapi score) becomes much bigger than the portion (Term score), and the results are similar to that of Okapi originally.

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 25 | 31 | 36 | 40 | 43 |
| Improvement | 5 | 5 | 8 | 8 | 4 |
| %Improvement | 25% | 19.2% | 28.6% | 25% | 10.3% |

**Table 3.** Results of combining term scores and Okapi scores.

The results are better than those of the first experiment, in particular for n=2 to 5. Note that the results for different n's correspond to different values of OW.

## 4.3. Experiment 3: Role of the rank of candidates

Analyzing the results of Experiment 2, which is somewhat weak for n=1, we found that the portion (OW * Okapi score) did not well solve the problem of bad rearrangement in Experiment 1. This is because, in many cases, the Okapi scores of the candidates of a given question are not distinguishing enough, e.g., they are just different from each other by some thousandths. In these cases, even with a large OW, it is the portion of term score that plays the main role in the combined scoring, and the system behaves similarly to that in the first experiment. This analysis has led us to giving a role in the scoring system to the rank of the candidates in the order returned by Okapi, by modifying formula (2) into:

**(3) Score of candidate[i] = RC[i] * Term score + OW * Okapi score**

where i is the rank of the candidate in the order returned by Okapi, and RC[i] is a corresponding coefficient (*RC* for *rank coefficient*). The problem now is how to find the best values of the RC vector.

There can be many choices here. The trivial case is RC=(1, 1, 1,..., 1), where (3) is identical to (2). If one notes that correct candidates are distributed mostly in the first ranks (n=1 to 6) (see C(n) in Table 1), one can think of an RC=(1, 1, 1, 1, 1, 1, 0.5, 0.5, 0.5, 0.5) for example. If we want to boost more the candidates of ranks 2 to 6, we can use RC=(1, 1.5, 1.5, 1.5, 1.5, 1.5, 0.5, 0.5, 0.5, 0.5). If we note the decrease of C(n), we can propose RC=(1, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1). If we consider the relative values of C(n) (a probabilistic view), we can think of RC=(1, 0.55, 0.25, 0.20, 0.45, 0.15, 0.05, 0.05, 0.20, 0.05)[2], etc. In general, one can construct RC values according to the forms a, b, and c in Figure 1. (The horizontal axis corresponds to ranks i from 1 to 10, the vertical axis to RC[i].) The form d gives bad results, because it boosts the score of low rank candidates (ranks 7 to 10).

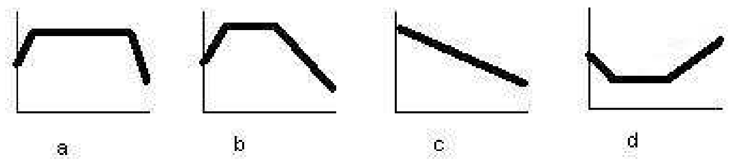

**Figure 1.** Possible distributions for constructing RC vectors.

[2] RC[i]=C(i)/C(1).

We have thus constructed about 50 values of RC, and run the system with these values and OW's from 0 to 60. The results continue to be better (Table 4). It is interesting to note that the best improvements for n=1 correspond mainly to the form a in Figure 1 (e.g., with RC=(1, 1.2, 1.2,..., 1.2, 0.1), RC=(1, 1.8, 1.8,..., 1.8, 0.1), RC=(1, 3, 3,...,3, 0)). For n=2, it is the form b, n=3, the form a; n=4, the form b; and n=5, all the forms a, b, and c.

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 26 | 33 | 37 | 43 | 43 |
| Improvement | 6 | 7 | 9 | 11 | 4 |
| %Improvement | 30% | 26.9% | 32.1% | 34.4% | 10.3% |

**Table 4.** Results with rank coefficients RC.

### 4.4. Experiment 4: Relationship between the question and the document via special terms

As formula (3) still cannot solve the problem of bad rearrangement, in this experiment, we test the speculation that if a candidate has been extracted from a document which does not contain any special term occurring in the question, this should be a "bad" candidate, and should be hence lowered in rank or even eliminated. This can be modeled by modifying (3) into:

**(4) Score of candidate[i] = TC * (RC[i] * Term score + OW * Okapi score)**

where *TC (term-related coefficient)* will be small if candidate[i] is "bad", and big if it is "good", e.g. TC=1 for the former case and TC=2 for the latter. We construct 20 such pairs of values of TC for training, e.g., (1, 1), (0, 1), (1, 1.5), (1 2), etc. The pair (1, 1) corresponds to the trivial case, where formula (4) is identical to (3). The pair (0, 1) corresponds to the extreme case of throwing away all bad candidates. This time the results look very good (Table 5). In particular, the best improvements for n=1 and 2 are made only with the pair (0, 1). Note that these improvements correspond to non-trivial values of OW and RC, showing that our arguments in preceding experiments are still valid. For n=3 and 4, the improvements are not best at the pair (0, 1), but yet very good at that pair.

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 30 | 38 | 41 | 43 | 44 |
| Improvement | 10 | 12 | 13 | 11 | 5 |
| %Improvement | 50% | 46.2% | 46.4% | 34.4% | 12.8% |

**Table 5.** Results with TC.

## 5. Improving the precision of the system with a concept hierarchy

In another effort to find a semantic characteristic of a document, we try to map the documents into a hierarchy of concepts. Each document corresponds to a set of concepts, and a concept corresponds to a set of documents. Building such a concept hierarchy 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 its creator. If no such classification existed, one can make use of techniques of building hierarchies of clusters (e.g. those summarized in Kowalski (1997)). However they are painstaking in programming and may not be very precise.

### 5.1. Building the concept hierarchy

We are lucky finding that the web page URL of a document can help here as the initial topical classification[3]. Although the URLs are complex, they contain a portion which can be used to construct

[3] Fortunately this information has been kept with almost all documents of the corpus, even with documents containing several web pages.

the concept hierarchy and the mapping. For example, below are the URLs of documents talking about the long distance (wired) phone service, its 'First Rate', and 'First Rate 24' plans, respectively. The portions in bold characters are useful portions:

- http://www.bell.ca/shop/application/commercewf?origin=noorigin.jsp&event=link(goto)&**content=/jsp/content/personal/catalog/phoneservices/long_distance/index.jsp**&REF=HP_PERS
- http://www.bell.ca/shop/application/commercewf?origin=noorigin.jsp&event=link(goto)&**content=/jsp/content/personal/catalog/phoneservices/long_distance/first_rate/index.jsp**
- http://www.bell.ca/shop/application/commercewf?origin=noorigin.jsp&event=link(goto)&**content=/jsp/content/personal/catalog/phoneservices/long_distance/first_rate_24/index.jsp**

After some string manipulations (implemented easily in Perl), including throwing away useless parts, like '/jsp/content/', 'catalog', 'index.jsp', etc., we obtain labels which can be used as the characterizing topic or concept of a document, e.g.:

```
Personal-Phone-LongDistance
Personal-Phone-LongDistance-FirstRate
Personal-Phone-LongDistance-FirstRate24
```

It is easy then to build a concept hierarchy from these labels: one which is a prefix of another will be its parent concept. After some manual editing for exceptional cases[4], we obtain a nice concept hierarchy and mapping between it and the document collection. Below is a small extracted portion of the hierarchy:

```
!Bellroot!
      Personal
            Personal-Phone
                  Personal-Phone-LongDistance
                        Personal-Phone-LongDistance-BasicRate
                        Personal-Phone-LongDistance-FirstRate
                        Personal-Phone-LongDistance-FirstRate24
                        Personal-Phone-LongDistance-FirstRateOverseas
                        ...
                  Personal-Phone-Lines
                  Personal-Phone-PhoneCards
                  ...
            Personal-Wireless
            Personal-Internet
      Business
            ...
      CustomerCare
            ...
```

### 5.2. Mapping from questions to documents via concepts

The use of the concept hierarchy in the QA system is 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, we can determine a small set of documents relevant to these concepts, and carry out the search of answers in this set.

[4] For documents containing several web pages, we use a concept common for all of them.

To map a question to the concept hierarchy, we postulate another assumption, *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". Except some exceptions[5], for every concept, we build a bag of words which make up the concept, e.g., for the concept `Personal-Phone-LongDistance-FirstRate`, the word bag is {"personal", "phone", "long", "distance", "first", "rate"}. We also build a small lexicon of (near) synonyms as mentioned above.

Now, a question will be analyzed into separate words, and we look for concepts whose word bags having elements in common with them. A concept is judged more relevant to a question if: (1) its word bag has more elements (in lemmatized form) in common with the question's words; (2) the (percentage) quotient of the subset in (1) over the entire word bag is higher; and (3) there are more occurrences of words in that subset in the question. For the example question above, here are the first 10 relevant concepts in descending order (the numbers following the concepts are their ranks):

```
Personal-Phone-LongDistance-FirstRate             1
Personal-Phone-LongDistance-FirstRateOverseas     2
Personal-Phone-LongDistance-FirstRate24           2
Business-Voice-LongDistance-CallingCard           4
Business-Voice-LongDistance-SavingsPlan           4
Business-Voice-LongDistance-PerCall               4
Personal-Phone-LongDistance-BasicRate             7
Personal-Phone-LongDistance                       8
Business-Voice-LongDistance                       8
Personal-Wireless-RatePlans                      10
```

From the relevant concept sets, it is straightforward to derive the relevant document set for a given question. The documents will also be ranked according to the order of the deriving concepts. (If a document is derived from several concepts, the highest rank will be used.)

The idea here is very similar to that in Experiment 4. Many concepts in fact coincide with a special term, e.g. 'First Rate', 'First Rate 24'. However, there are still many concepts which are not special terms, e.g. 'phone', 'wireless', 'long distance', etc.

### 5.3. Experiment 5: Improving IR precision via concepts

To determine whether the list of documents derived for a question as described above makes some difference, we use a formula similar to (4):

**(5) Score of candidate[i] = CC * (RC[i] * Term score + OW * Okapi score)**

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 28 | 36 | 40 | 41 | 42 |
| Improvement | 8 | 10 | 12 | 9 | 3 |
| %Improvement | 40% | 38.5% | 42.9% | 28.1% | 7.7% |

**Table 6.** Results with CC.

where *CC* (for *concept-related coefficient*) is a coefficient depending on the document that provides the candidate[i]. 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 small, e.g. 0. We run the system with 6 different values of the CC vector, and obtain results

[5] For example, we did not build word bags for the concepts 'Business' and 'Personal', because words like 'business' and 'personal' or 'person' are not highly suggestive of the corresponding concepts. Moreover, the number of documents relevant to these concepts is too large for any meaningful selection.

shown in Table 6. The results are better than when the system does not use CC (see Table 4), but less good than when it uses TC (Table 5). This suggests us to combine TC and CC in the next experiment.

### 5.4. Experiment 6: Combining term-related scoring (TC) and concept-related scoring (CC)

We make a simple combination of TC and CC as below:

**(6) Score of candidate[i] = (TC + CC) * (RC[i] * Term score + OW * Okapi score)**

This time, the results (Table 7) is slightly better than when using TC alone (Table 5). This may be explained by the fact that TC and CC are conceptually close methods, and derive similar candidate rearrangements.

| n | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Q(n) of Okapi | 20 | 26 | 28 | 32 | 39 |
| Q(n) of system | 30 | 39 | 43 | 44 | 44 |
| Improvement | 10 | 13 | 15 | 12 | 5 |
| %Improvement | 50% | 50% | 53.6% | 37.5% | 12.8% |

**Table 7.** Results with TC and CC combined.

## 6. Final tests

Finally, we perform the final tests with the optimal values of CC, TC, RC, and OW on the test question set (40 questions). By curiosity, we do two final tests, one with formula (4), i.e. no CC, and one with the last formula (6), to see whether (6) is better than (4) over the new question set.

The results in Table 8 show that the system has performed very good improvements with n=1 and n=2, but not as well with n=3 to 5. This can be explained by the fact that, in the Okapi candidate set, most of the correct candidates from ranks 6 to 10 correspond to questions that have already a correct candidate in ranks 1 to 5: only 25-22=3 more questions will obtain a correct candidate if one extends the number of candidates from 5 to 10. Therefore those correct candidates from ranks 6 to 10 did not contribute much in the process of pushing good candidates to the first ranks, yielding low improvements for n=3 to 5. Finally, results of formula (6) are uniformly better than those of formula (4), conforming to our prediction.

| n | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| C(n) of Okapi | 10 | 6 | 7 | 2 | 3 | 2 | 3 | 2 | 1 | 0 |
| Q(n) of Okapi | 10 | 14 | 19 | 20 | 22 | 22 | 24 | 25 | 25 | 25 |
| %Q(n) of Okapi | 25% | 35% | 47.5% | 50% | 55% | 55% | 60% | 62.5% | 62.5% | 62.5% |
| Q(n) of system using formula (4) (no CC) | 15 | 19 | 22 | 23 | 23 | | | | | |
| Improvement to Okapi | 5 | 5 | 3 | 3 | 1 | | | | | |
| %Improvement to Okapi | 50% | 36% | 16% | 15% | 5% | | | | | |
| Q(n) of system using formula (6) | 16 | 21 | 23 | 24 | 24 | | | | | |
| Improvement to Okapi | 6 | 7 | 4 | 4 | 2 | | | | | |
| %Improvement to Okapi | 60% | 50% | 21% | 20% | 9% | | | | | |

**Table 8.** Results of the final test.

## 7. Discussions and conclusions

In this work, we have made considerable improvements on the precision of the IR module of the QA system. Special terms, constituted by the service names of Bell Canada, and the concept hierarchy, built from the original document classification of Bell Canada website, play a major role in these improvements. One can wonder why such a performance was not achieved by the IR engine (Okapi here) itself, even though these special terms had been entered into the engine as indices (see footnote 1). The reason may be that the engine treated these terms equally to other terms. By giving high scores to special terms, together with designing the coefficients OW, RC, TC, and CC, we have made the system more sensitive to the working term set of the application.

The fact that we had convenience in extracting special terms which occur in capital letters, and building the concept hierarchy from the document's URLs, does not diminish the generality of the approach. The main idea here is to find some kind of information which can efficiently characterize the relevance of a candidate to the corresponding question. This kind of information can be the terminological set most used in the interested task. It can be constructed by hand or (semi-) automatically using different term extraction techniques. As for the concept hierarchy, it is reasonable to hypothesize that some classification exists for a domain-specific document set. However, our work is not about how to construct such a concept hierarchy, but rather how to apply it in finding correct answers.

## Acknowledgements

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

## References

Brown, J., Burton, R. (1975). Multiple representations of knowledge for tutorial reasoning. In Bobrow & Collins (Eds), *Representation and Understanding*, pp. 311-350. Academic Press, New York.

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

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of American Society of Information Science, 41*, 391-407.

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, 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.

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.

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.

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