Using Terminology and a Concept Hierarchy for Restricted-Domain Question-Answering

Hai Doan-Nguyen and Leila Kosseim

Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, H3G 1M8, Canada
haidoan@cse.concordia.ca, kosseim@cse.concordia.ca

Abstract. This paper discusses an important characteristic of restricted-domain question-answering (RDQA): the issue of low precision in finding good answers. We propose methods for improving precision using domain-specific terminology and concept hierarchy 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 information retrieval engine in a two-level search strategy. The methods proposed here have been applied to a real QA system for a large telecommunication company, yielding significant improvements in precision.

1 Introduction

This paper presents our research in the development of a question-answering (QA) system for a restricted domain. The system's goal is to reply to customer's questions on services offered by Bell Canada, a major Canadian telecommunication corporation. Although grounded within a specific context, we try to reveal general problems and develop a general methodology for restricted-domain QA (RDQA).

Although work in RDQA dates back to the early years of Artificial Intelligence, the domain has only recently regained interest in the research community. RDQA performs QA on a specific domain and often uses document collections restricted in subject and volume. Often integrated in real-world applications, RDQA, especially systems working on free text corpora (rather than on structured databases), provides many interesting challenges to natural language engineering. Real questions have no limit on form, style, category, and complexity. In addition, a RDQA system often has to deal with the problem of low precision in finding a correct answer for a given question.

In this paper, we discuss the main characteristics of RDQA, and present a series of methods to improve the precision performance of our system. These methods were not developed specifically for our project at hand but always considered in a general perspective of RDQA.

© *A. Gelbukh (Editor)*
Advances in Natural Language Processing
Research in Computing Science 18, 2006, pp. 183-194

2 Restricted-Domain QA (RDQA)

In the early days of Artificial Intelligence, work in QA typically addressed restricted-domain systems working over databases (e.g., [1] and [2]). However, at the end of the last decade, interest in QA has been stirred up by the availability of huge volumes of electronic documents, especially the WWW, and of powerful search engines. Within this context, researchers have been most attracted to open-domain QA, driven by the TREC and DARPA initiatives. Most work has been performed on finding precise and short answers to factoid questions. However, as QA is applied to more practical tasks, it has become apparent that this orientation cannot satisfy the requirements of practical applications. It is RDQA, re-emerging recently, that brings out a set of new directions to the domain. Expected to carry out QA principally in real-life contexts, particularly in industrial and commercial applications, RDQA has to deal with real and very difficult problems. It is no surprise that RDQA is regaining attention these days at the research level, as evidenced by, e.g., [3–5]. Some typical work on RDQA include the LILOG project that answers questions on tourist information [6], the ExtrAns system working on Unix manuals [7], WEBCOOP also on tourist information [8], and the system of [9] working on the telecommunications domain.

Techniques developed for open-domain QA, particularly for TREC competitions, are not that helpful in RDQA. Indeed, RDQA has several characteristics that make it different from open-domain QA:

Restricted Document Collection In RDQA, the document collection is typically restricted in subject and in volume. By definition, the domain of knowledge and information of interest is predefined, and often very narrow. The working document collection is normally much smaller in size than that of open-domain QA systems. In addition, it is often homogeneous in style and structure.

Scarcity of Answer Sources A consequence of a restricted document collection is the scarcity of answer sources. Redundancy of answer candidates for a given question is exploited extensively in open-domain QA systems as one principal method to determine the right answer. The situation is contrary in RDQA. As the documents often come from only a few sources, they do not likely have repeated contents. Information about a specific issue is typically referred to in only a few areas of the documents, and the system will not have a large retrieval set abundant of good candidates for selection. [10] shows that the performance of a system depends greatly on the redundancy of answer occurrences in the document collection. 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.) Scarcity of correct answers explains why low performance on precision is a general concern for RDQA systems.

Domain Specific Terminology A RDQA system normally has to work with domain-specific terms and meaning. As these will not be found in a general dictionary or thesaurus, lexical and semantic techniques based on the use of these resources (consider the wide use of WordNet in open-domain QA) may not apply well here. Instead we need specialized knowledge, such as a technical term dictionary, an ontology of entities in the domain, etc. Nonetheless, building these resources from scratch may be costly and problematic, because it often requires domain expertise.

On the other hand, as in any specialized domain, word sense disambiguation may be a smaller issue in RDQA. Most "important" words will only have a few possible senses. For instance, the word *card* in our corpus seems to have only one sense (telephone card), compared to 11 senses in WordNet 2.1.

Complex Questions If a QA system is to be used for a real application, e.g. answering questions from clients of a company, it should accept arbitrary questions, of various forms and styles. Contrarily, current open-domain QA systems generally suppose 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 ...). Complexity of questions can be studied at different levels: *representational level*: syntax, number of sentences, style, ...; *semantic level*: entities in the question and their relationships; *logical level*: hypotheses, presuppositions, inferences, ..., and *intentional level:* expectations, objectives, ... For example, let's consider a question from our corpus:

> *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?*

This is a rather complex question at the representational level, because it is made up of two sentences, both, syntactically complex. Semantically, it asserts something about the *First Rate Plan*, and asks for another plan for long distance telephone calls. At the logical level, one can discover that the client has presupposed (correctly) that *First Rate Plan* is a long distance telephone call plan, that this plan is not economical during the day. At the intentional level, she wants to know whether there is another similar but less expensive plan, and expects to receive various information about it (how to register, its price structure ...).

Complex Answers Answers in real-life QA cannot be restricted to simple phrases or short snippets. They should contain as much information as needed in order to answer the true intention of the question. This may imply clarifying the context of the problem posed in the question, explaining different situations, providing justifications and inferences, giving instructions or suggestions, ... For the example above, it would be a business disaster for the company if the QA system only returned a specific plan name, e.g. *"First Rate 24 !"* with no further

explanations or details. Finding an answer that contains the necessary information (*completeness*), but only the necessary information (*conciseness*) is far more difficult with non-factual questions where the answer is not simply defined as a grammatical constituent. An ideal answer may be as follows:

> **A:** *First Rate 24*
> *$4.95/month*
> *If most of your long distance calling is within Canada or to the United States during the day or if you want convenience of calling anytime during the day or night, then the First Rate 24 hours plan may be a good option for you. With the First Rate 24 hours long distance plan you will receive the following benefits.*
> *- Now you never have to worry about the time when you're calling friends, relatives, or following up on an important business call.*
> *- For a low fee of $4.95 a month, you can call anywhere in Canada for just 10 cents a minute, and to the U.S. for just 20 cents a minute, 24 hours a day, everyday.*
> *- You'll also be able to take advantage of our 24 hour flat overseas rates on direct dialled calls to our 28 most popular overseas calling destinations.*

Evaluation Finally, evaluation of a RDQA system cannot be performed using the same techniques as in open-domain QA. In open-domain QA, answers are often very short, hence one can look for a specific pattern in the answers and automate the evaluation process, in order to compute various measures (e.g. MRR, confidence-weighted score, etc). In the case of complex questions and answers, a more complete evaluation is necessary. As an example, [11] present a novel and comprehensive way to evaluate a system as a whole, but approaches inspired by the evaluation of automatic summarization systems can also be used.

3 RDQA for Bell Canada

3.1 The Corpus

In our project, the document collection was derived from Bell Canada's website (`www.bell.ca`) in 2003. It contains descriptions of the company's wide-range products and services in telecommunication (telephone, wireless, Internet, Web, etc.). As the documents were in various formats (e.g. HTML, PDF) they were saved as plain text, sacrificing some important formatting cues. The collection comprises more than 220 documents, for a total of about 560K characters.

The available question set has 140 questions, whose form and style vary freely. Most questions are composed of one sentence, but some are composed of several sentences. The average length of the 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?*

3.2 Initial Evaluation: Using a naïve QA system

Although our collection was not very large, it was not so small either so that a strategy of searching 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. The well-known generic IR engine OKAPI (`www.soi.city.ac.uk/~andym/OKAPI-PACK/`, also [12]) helped us to do both steps. For each question, input as such, 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[1]. OKAPI is thus more than just a traditional IR engine; it can be regarded as a naïve QA system returning paragraphs.

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 3.2 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 correct answer was very rare. C(n) is the number of all 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

[1] A paragraph is a block of text separated by two newline characters. 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.

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 3.2 also gives OKAPI's precision on the test question set.

Table 1. Performance of OKAPI on the training question set (80 questions), and the test question set (60 questions).

n	1	2	3	4	5	6	7	8	9	10
Training Set										
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%
Test Set										
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%

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 a more practical scenario 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 considered 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 other well-known scoring such as mean reciprocal rank (MRR) or confidence-weighted score.

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 information. However, only 2/3 of the latter (that is 10% of all) looked difficult to be completed automatically. Generating a more concise answer from a good candidate (an extracted paragraph) therefore seemed less important than improving the precision of the IR module. We therefore concentrated on how to improve Q(n) for n= 1 to 5.

4 Improving Precision

The first obvious approach to improve the precision performance of the system is to use a better IR engine or tuning the current one to the specific domain and

task, e.g. by adjusting the parameters, modifying the weighting formulas of the engine. However, this approach is neither general (an engine may work well with one application but not with another), nor interesting at the theoretical level.

The second approach consists in improving the results returned by the IR engine. One main direction is re-ranking the candidates, 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.

While implementing re-ranking methods, we found that domain-specific semantic information could be used to filter documents according to their relevance to a given question. This led us to another approach: concept-based two-level search. In the following, we will present our development of a series of strategies for precision improvement and their results.

4.1 Re-ranking Candidates

In this approach, we experimented with two methods of re-ranking: one with a strongly specific terminological set, and one with a concept hierarchy allowing a good document characterization.

Re-ranking using terminology 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 such terms occurred very often in almost every document and question, and a specific service was often presented or mentioned in only one or a few documents, making these terms very discriminating. Luckily, these *domain terms* occur typically in capital letters in the corpus, and could easily be extracted automatically. After a manual filtering, we obtained more than 450 domain terms.

We therefore designed a new scoring method which increases the score of candidates containing occurrences of domain terms found in the corresponding question. Of course, the system also combine Okapi's scoring in the computation. Due to space limit, we refer readers to [13] and [14] for the details of the experiment.

The new terminology-based scoring gave very encouraging improvements on the training set, but just modest results when running with optimal training parameters on the test set (see Table 4.1). What encouraged us here was that improvement was shown to be possible; we just had to look for better characterizing information. The scoring system devised here was also used as the basic model for the next set of experiments.

Re-ranking using a concept hierarchy To better re-rank the candidates, we now focused on how to guess which documents most probably provide a good

Table 2. Best results of the terminology-based scoring on the training set, and results of applying it with optimal training parameters on the test set.
Note: $\Delta Q(n)$ = System's Q(n) - OKAPI's Q(n); $\%\Delta Q(n) = \frac{\Delta Q(n)}{okapi's Q(n)}$.

n	1	2	3	4	5
Training Set					
Q(n)	30	40	42	43	44
$\Delta Q(n)$	10	14	14	11	5
$\%\Delta Q(n)$	50%	53.8%	50%	34.4%	12.8%
Test Set					
Q(n)	22	29	32	33	34
$\Delta Q(n)$	4	6	4	4	2
$\%\Delta Q(n)$	22.2%	26.1%	14.3%	13.8%	6.3%

candidate for a given question. For this purpose, we tried to map the documents into a system of concepts. Each document refers to a set of concepts, and a concept is discussed in a set of documents. Building such a concept system is 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. [15]). In our case, a concept hierarchy and the mapping between it and the document collection was easily built based on the original document classification of Bell Canada. Below is a small excerpt from the hierarchy:

```
Bellall
    Personal
        Personal-Phone
            Personal-Phone-LongDistance
                Personal-Phone-LongDistance-BasicRate
                Personal-Phone-LongDistance-FirstRate
```

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.* To be precise, we should measure the relevance of a concept to a given question. For example, the concepts `Personal-Phone-LongDistance-FirstRate` and `Personal-Phone-LongDistance` are highly relevant to the question *"It seems that the First Rate Plan is only good if most of my calls... "* mentioned in section 2. From that measure, it is easy to compute the relevance of a document to a question using the concept-documents mapping. If we keep only the concepts and documents having a positive relevance measure with a question, we have a question-concepts and a question-documents mapping, respectively.

Now it seems that we can better re-arrange the candidates with a new kind of information: a candidate will be ranked higher if its containing document is more relevant to the question. Details of the implementation of this strategy,

including how to measure concept-question and document-question relevance, are given in [14]. The uniformly good improvements (see Table 4.1) show that the approach is appropriate and effective.

Table 3. Best results of the concept-based scoring on the training set, and results of applying it with optimal training parameters on the test set.

n	1	2	3	4	5
Training Set					
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%
Test Set					
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%

4.2 Concept-Based Two-Level Search

As the concept-based mappings in the previous section seem 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. In the previous experiments, the IR engine searches over the entire document collection. Now search will be carried out in two levels: at the *semantic level*, the system determines a subset of most promising documents for each question; at the *IR level*, the engine just searches in this subset. We hoped that search could achieve higher precision in working with a much smaller document set, which usually contains no more than 20 documents in our case. As we use the concept-based mappings at the semantic level, we will call this strategy *conceptual-mapping-then-IR*.

Conceptual Mapping then Okapi We first experimented our two-level search strategy with OKAPI, where indexing and search were performed for each question on its relevant document subset. The results were not better than when OKAPI worked with the entire document collection. We then applied the scoring system devised in the previous section to rearrange the candidate lists. Although the results on the training set were generally better than those of the previous section, results on the test set were worse, which led to an unfavourable conclusion for this method. Details of the experiments here can be found in [14].

Using a new IR engine The precision of the conceptual mappings was good, but the performance of the two-level system using OKAPI was not convincing. This may be because OKAPI cannot work well with very small document sets.

We therefore implemented another IR engine, which is simple but adapted for working with small document sets. It tries to identify an excerpt in a given document that is most likely to answer the input question. Details of the system is presented in [14]. Note that if the relevant document subset for a given question is empty, the system takes the candidates proposed by OKAPI as results (i.e. *okapi as last resort*).

Also in this implementation, if the searched document is "small", i.e. contains less than 2000 characters, the system simply takes the entire document as the candidate. This 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 judgment in the scenario of a semi-automatic system. We call this case *"candidate expansion"*, because whatever a candidate may be, it is expanded to the whole document.

The results (Table 4.2) show that except for the case of n=1 in the test set, the new system performs well for precision. What is interesting here is that although simple, the engine is quite effective, because it does searching on a well selected and very small document subset.

Table 4. Best results of the new IR engine on the training set, and results of this method (with optimal training parameters) on the test set.

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

Expanding 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 approach of improving precision by improving the results returned by the IR engine. To be fair, it is necessary to see how precision will be improved if this candidate expansion is used in other experiments. We thus performed two further experiments. In the first one, any candidates returned by OKAPI (cf. Table 3.2) which came from a document of less than 2000 characters were expanded into the entire document. In the second experiment, we similarly expanded candidates returned by *conceptual-*

mapping-then-okapi. Results showed that improvements are not as good as those obtained by other methods. See details in [14]. The two experiments suggest that expanding 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 expansion with re-ranking.

5 Conclusion and Future Work

RDQA, working on small document collections and restricted subjects, seems to be no less difficult a task than open-domain QA. Due to candidate scarcity, the precision 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 of the IR module. They include the use of domain-specific terminology and concept hierarchy 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 are a common characteristic of RDQA systems. Being aware of this, one can design an appropriate system which is more tolerant to answer size to achieve a higher precision, and to avoid the need of expanding a short but insufficient answer into a complete one.

Good improvements achieved when applying our methods to a QA system for Bell Canada shows that these methods are applicable and effective. Although the experiments were grounded in a real-world situation, we have tried to build the strategies as general and as modular as possible in order to develop a general methodology for the task of RDQA.

Many problems on the precision performance of a RDQA system have been suggested in this paper. First, there is the problem of how to build a *good* answer. We have worked with finding correct answers. However, a correct answer may not be *good*, because it may be vague, lengthy, superfluous, etc. The ultimate goal of QA is to find answers that satisfy the questioner's needs rather than just correct ones, and we are currently working on this problem. Second, there is the problem of how to analyze an arbitrary question into semantic and logical parts (entities mentioned in the question and their relationships, pre-suppositions, problem context, question focus, etc.) so that the system has a better understanding of what is being asked. Third, the general problem in Natural Language Processing, of analysing free text at any level. Here, work performed in automatic topic detection and content segmentation may be helpful to identify relevant answers. Certainly, these also constitute problems of open-domain QA if one wants to explore the domain further than the typical factoid questions.

Acknowledgments This project was funded by Bell University Laboratories (BUL) and the Canada Natural Science and Engineering Research Council (NSERC). The authors would also like to thank to anonymous referees for their comments.

References

1. Green, B.F., Wolf, A.K., Chomsky, C., Laughery, K.: Baseball: An Automatic Question Answerer. In: Proceedings of the Western Joint Computer Conference. (1961) 219–224
2. Woods, W.: Progress in Natural Language Understanding: An Application to Lunar Geology. In: AFIPS Conference Proceeding. Volume 42. (1973) 441–450
3. Mollá, D., Vicedo, J.L., eds.: Workshop on Question Answering in Restricted Domains - ACL-2004, Barcelona, Spain (2004)
4. Mollá, D., Vicedo, J.L., eds.: AAAI-05 Workshop on Question Answering in Restricted Domains, Pittsburg, Pennsylvania (2005) to appear.
5. Mollá, D., Vicedo, J.L., eds.: Special Issue of Computational Linguistics on Question Answering in Restricted Domains. (2005) to appear.
6. Herzog, O., Rollinger, C.R., eds.: Text Understanding in LILOG, Integrating Computational Linguistics and Artificial Intelligence, Final Report on the IBM Germany LILOG-Project. In Herzog, O., Rollinger, C.R., eds.: Text Understanding in LILOG. Volume 546 of Lecture Notes in Computer Science., Springer (1991)
7. Mollá, D., Berri, J., Hess, M.: A real world implementation of answer extraction. In: Proceedings of the 9th International Workshop on Database and Expert Systems, Workshop: Natural Language and Information Systems (NLIS-98), Vienna (1998)
8. Benamara, F., Saint-Dizier, P.: Advanced Relaxation for Cooperative Question Answering. In: New Directions in Question Answering. (2004) 263–274
9. Oroumchian, F., Darrudi, E., Ofoghi, B.: Knowledge-Based Question Answering with Human Plausible Reasoning. In: Proceedings of the 5th International Conference on Recent Advances in Soft Computing. (2004)
10. Light, M., Mann, G., Riloff, E., Breck, E.: Analyses for Elucidating Current Question Answering Technology. Natural Language Engineering **7** (2001)
11. Diekema, A.R., Yilmazel, O., Liddy, E.D.: Evaluation of Restricted Domain Question-Answering Systems. In: Proceedings of the Association of Computational Linguistics 2004 Workshop on Question Answering in Restricted Domains (ACL-2004), Barcelona, Spain (2004)
12. Robertson, S.E., Walker, S., Jones, S., Hancock-Beaulieu, M.M., Gatford, M.: OKAPI at TREC-3. In D.K.Hartman, ed.: Overview of the Third TExt Retrieval Conference (TREC-3), Gaithersburg, NIST (1995) 109–130
13. Doan-Nguyen, H., Kosseim, L.: Improving the Precision of a Closed-Domain Question-Answering System with Semantic Information, Avignon, France, RIAO-2004 (2004)
14. Doan-Nguyen, H., Kosseim, L.: The Problem of Precision in Restricted-Domain Question-Answering. Some Proposed Methods of Improvement, Barcelona, Spain, ACL (2004)
15. Kowalski, G.: Information Retrieval Systems – Theory and Implementation. Kluwer Academic Publishers, Boston/Dordrecht/London (1997)