MAYA: A Fast Question-answering System Based On A Predictive Answer Indexer*

Harksoo Kim, Kyungsun Kim
Dept. of Computer Science,
Sogang University
1 Sinsu-Dong, Mapo-Gu, Seoul,
121-742, Korea
{ hskim, kksun }
@nlpzodiac.sogang.ac.kr

Gary Geunbae Lee
Dept. of Computer Science
and Engineering,
Pohang University of
Science and Technology
San 31, Hyoja-Dong,
Pohang, 790-784, Korea
gblee@postech.ac.kr
(Currently Visiting CSLI Stanford University)

Jungyun Seo
Dept. of Computer Science,
Sogang University
1 Sinsu-Dong, Mapo-Gu,
Seoul, 121-742, Korea
seojy@ccs.sogang.ac.kr

Abstract

We propose a Question-answering (QA) system in Korean that uses a predictive answer indexer. The predictive answer indexer, first, extracts all answer candidates in a document in indexing time. Then, it gives scores to the adjacent content words that are closely related with each answer candidate. Next, it stores the weighted content words with each candidate into a database. Using this technique, along with a complementary analysis of questions, the proposed QA system can save response time because it is not necessary for the QA system to extract answer candidates with scores on retrieval time. If the QA system is combined with a traditional Information Retrieval system, it can improve the document retrieval precision for closed-class questions after minimum loss of retrieval time.

1 Introduction

Information Retrieval (IR) systems have been applied successfully to a large scale of search area in which indexing and searching speed is important. Unfortunately, they return a large amount of documents that include indexing terms in a user's query. Hence, the user should carefully look over the whole text in order to find a short phrase that precisely answers his/her question.

Question-answering (QA), an area of IR, is attracting more attention, as shown in the proceedings of AAAI (AAAI, 1999) and TREC (TREC, *http://trec.nist.gov/overview.html*). A QA system searches a large collection of texts, and filters out inadequate phrases or sentences within the texts. By using the QA system, a user can promptly approach to his/her answer phrases without troublesome tasks. However, most of the current QA systems (Ferret et al., 1999; Hull, 1999; Srihari and Li, 1999; Prager et al., 2000) have two problems as follows:

- It cannot correctly respond to all of the users' questions. It can answer the questions that are included in the pre-defined categories such as *person*, *date*, *time*, and etc.
- It requires more indexing or searching time than traditional IR systems do because it needs a deep linguistic knowledge such as syntactic or semantic roles of words.

To solve the problems, we propose a QA system using a predictive answer indexer - MAYA (MAke Your Answer). We can easily add new categories to MAYA by only supplementing domain dictionaries and rules. We do not have to revise the searching engine of MAYA because the indexer is designed as a separate component that extracts candidate answers. In addition, a user can promptly obtain answer phrases on retrieval time because MAYA indexes answer candidates in advance.

* This research was partly supported by BK21 program of Ministry of Education and Technology Excellency Program of Ministry of Information and Telecommunications.

Most of the previous approaches in IR have been focused on the method to efficiently represent terms in a document because they want to index and search a large amount of data in a short time (Salton et al., 1983; Salton and McGill, 1983; Salton 1989). These approaches have been applied successfully to the commercial search engines (e.g. *http://www.altavista.com*) in World Wide Web (WWW). However, in a real sense of information retrieval rather than document retrieval, a user still needs to find an answer phrase within the vast amount of the retrieved documents although he/she can promptly find the relevant documents by using these engines. Recently, several QA systems are proposed to avoid the unnecessary answer finding efforts (Ferret et al., 1999; Hull, 1999; Moldovan et al. 1999; Prager et al., 1999; Srihari and Li, 1999).

Recent researches have combined the strengths between a traditional IR system and a QA system (Prager et al., 2000; Prager et al., 1999; Srihari and Li, 1999). Most of the combined systems access a huge amount of electronic information by using IR techniques, and they improve precision rates by using QA techniques. In detail, they retrieve a large amount of documents that are relevant to a user's query by using a well-known TF·IDF. Then, they extract answer candidates within the documents, and filter out the candidates by using an expected answer type and some rules on the retrieval time. Although they have been based on shallow NLP techniques (Sparck-Jones, 1999), they consume much longer retrieval time than traditional IR systems do because of the addictive efforts mentioned above. To save retrieval time, MAYA extracts answer candidates, and computes the scores of the candidates on indexing time. On retrieval time, it just calculates the similarities between a user's query and the candidates. As a result, it can minimize the retrieval time.

This paper is organized as follows. In Section 2, we review the previous works of the QA systems. In Section 3, we describe the applied NLP techniques, and present our system. In Section 4, we analyze the result of our experiments. Finally, we draw conclusions in Section 5.

2 Previous Works

The current QA approaches can be classified into two groups; text-snippet extraction systems and noun-phrase extraction systems (also called closed-class QA) (Vicedo and Ferrández, 2000).

The text-snippet extraction approaches are based on locating and extracting the most relevant sentences or paragraphs to the query by assuming that this text will probably contain the correct answer to the query. These approaches have been the most commonly used by participants in last TREC QA Track (Ferret et al., 1999; Hull, 1999; Moldovan et al., 1999; Prager et al., 1999; Srihari and Li, 1999). ExtrAns (Berri et al., 1998) is a representative QA system in the text-snippet extraction approaches. The system locates the phrases in a document from which a user can infer an answer. However, it is difficult for the system to be converted into other domains because the system uses syntactic and semantic information that only covers a very limited domain (Vicedo and Ferrández, 2000).

The noun-phrase extraction approaches are based on finding concrete information, mainly noun phrases, requested by users' closed-class questions. A closed-class question is a question stated in natural language, which assumes some definite answer typified by a noun phrase rather than a procedural answer. MURAX (Kupiec, 1993) is one of the noun-phrase extraction systems. MURAX uses modules for the shallow linguistic analysis: a Part-Of-Speech (POS) tagger and finite-state recognizer for matching lexico-syntactic pattern. The finite-state recognizer decides users' expectations and filters out various answer hypotheses. For example, the answers to questions beginning with the word *Who* are likely to be people's name. Some QA systems participating in Text REtrieval Conference (TREC) use a shallow linguistic knowledge and start from similar approaches as used in MURAX (Hull, 1999; Vicedo and Ferrández, 2000). These QA systems use specialized shallow parsers to identify the asking point (*who*, *what*, *when*, *where*, etc). However, these QA systems take a long response time because they apply some rules to each sentence including answer candidates and give each answer a score on retrieval time.

MAYA uses shallow linguistic information such as a POS tagger, a lexico-syntactic parser similar to finite-state recognizer in MURAX and

a Named Entity (NE) recognizer based on dictionaries. However, MAYA returns answer phrases in very short time compared with those previous systems because the system extracts answer candidates and gives each answer a score using pre-defined rules on indexing time.

3 MAYA Q/A approach

MAYA has been designed as a separate component that interfaces with a traditional IR system. In other words, it can be run without IR system. It consists of two engines; an indexing engine and a searching engine.

The indexing engine first extracts all answer candidates from collected documents. For answer extraction, it uses the NE recognizer based on dictionaries and the finite-state automata. Then, it gives scores to the terms that surround each candidate. Next, it stores each candidate and the surrounding terms with scores in Index DataBase (DB). For example, if *n* surrounding terms affects a candidate, *n* pairs of the candidate and terms are stored into DB with *n* scores. As shown in Figure 1, the indexing engine keeps separate index DBs that are classified into pre-defined semantic categories (i.e. users' asking points or question types).

The searching engine identifies a user's asking point, and selects an index DB that includes answer candidates of his/her query. Then, it calculates similarities between terms of his/her query and the terms surrounding the candidates. The similarities are based on p-Norm model (Salton et al., 1983). Next, it ranks the candidates according to the similarities.

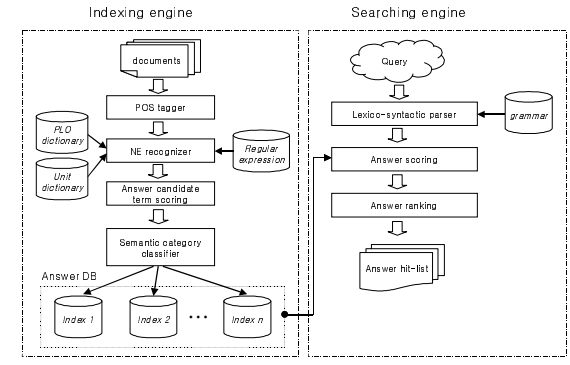

Figure 1. A basic architecture of the QA engines

Figure 2 shows a total architecture of MAYA that combines with a traditional IR system. As shown in Figure 2, the total system has two index DBs. One is for the IR system that retrieves relevant documents, and the other is for MAYA that extracts relevant answer phrases.

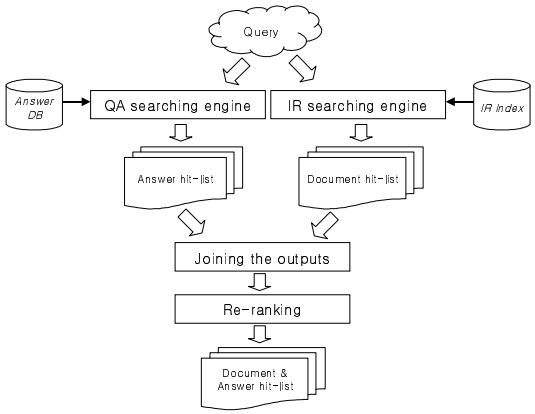

Figure 2. A total architecture of the combined MAYA system

3.1 Predictive Answer indexing

The answer indexing phase can be separated in 2 stages; Answer-finding and Term-scoring. For answer-finding, we classify users' asking points into 14 semantic categories; *person*, *country*, *address*, *organization*, *telephone number*, *email address*, *homepage Uniform Resource Locator* (*URL*), *the number of people*, *physical number*, *the number of abstract things*, *rate*, *price*, *date*, and *time*. We think that the 14 semantic categories are frequently questioned in general IR systems. To extract answer candidates belonging to each category from documents, the indexing engine uses a POS tagger and a NE recognizer. The NE recognizer makes use of two dictionaries and a pattern matcher. One of the dictionaries, which is called PLO dictionary (487,782 entries), contains the names of people, countries, cities, and organizations. The other dictionary, called unit dictionary (430 entries), contains the units of length (e.g. *cm*, *m*, *km*), the units of weight (e.g. *mg*, *g*, *kg*), and others. After looking up the dictionaries, the NE recognizer assigns a semantic category to each answer candidate after disambiguation using POS tagging. For example, the NE recognizer extracts 4 answer candidates annotated with 4 semantic categories in the sentence, "*야후코리아 (대표 염진섭 www.yahoo.co.kr)는 무료 이메일 용량을 6 메가로 늘렸다.* (*Yahoo Korea (CEO Jinsup Yeom www.yahoo.co.kr) expanded*

the size of the storage for free email service to 6 mega-bytes.)". *야후코리아* (*Yahoo Korea*) belongs to *organization*, and *염진섭*(*Jinsup Yeom*) is *person*. *www.yahoo.co.kr* means *homepage URL*, and *6 메가*(*6 mega-bytes*) is *physical number*. Complex lexical candidates such as *www.yahoo.co.kr* are extracted by the pattern matcher. The pattern matcher extracts formed answers such as *telephone number*, *email address*, and *homepage URL*. The patterns are described as regular expressions. For example, *Homepage URL* satisfies the following regular expressions:

- ^(http://)[_A-Za-z0-9\-]+(\.[_A-Za-z0-9\-]+)+(/[_~A-Za-z0-9\-\.]+)*$
- ^[0-9]{3}(\.[0-9]{3})(\.[0-9]{2,}){2}(/[_~A-Za-z0-9\-\.]{2,})*$
- ^[0-9]*[_A-Za-z\-]{1,}[_A-Za-z0-9\-]+(\.[_A-Za-7z0-9\-]{2,}){2,}(/[_~A-Za-z0-9\-\.]{2,})*$

In the next stage, the indexing engine gives scores to content words within a context window that occur with answer candidates. The maximum size of the context window is 3 sentences; a previous sentence, a current sentence, and a next sentence. The window size can be dynamically changed. When the indexing engine decides the window size, it checks whether neighboring sentences have anaphora or lexical chains.

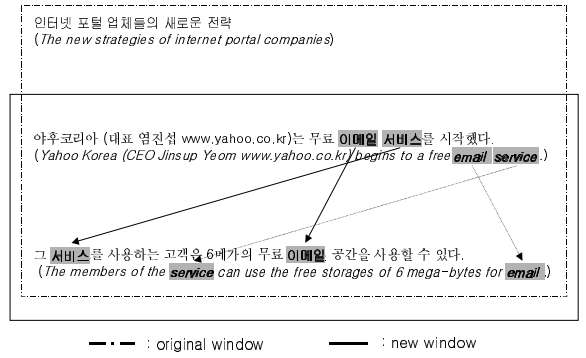

Figure 3. An example with the adjusted window size

If the next sentence has anaphors or lexical chains of the current sentence and the current sentence does not have anaphors or lexical chains of the previous sentence, the indexing engine sets the window size as 2. Unless neighboring sentences have anaphors or lexical chains, the window size is 1. Figure 3 shows an example in which the window size is adjusted.

The scores of the content words indicate the magnitude of influences that each content word causes to answer candidates. For example, when *www.yahoo.co.kr* is an answer candidate in the sentence, "*야후코리아(www.yahoo.co.kr)가 새로운 서비스를 시작한다.* (*Yahoo Korea (www.yahoo.co.kr) starts a new service.*)", *야후코리아*(*Yahoo Korea*) has the higher score than *서비스*(*service*) because it has much more strong clue to www.yahoo.co.kr. We call the score a term score. The indexing engine assigns term scores to content words according to 5 scoring features described below.

- POS: the part-of-speech of a content word. The indexing engine gives 2 points to each content word annotated with a proper noun tag and gives 1 point to each content word annotated with other tags such as noun, number, and etc. For example, *야후코리아* (*Yahoo Korea*) obtains 2 points, and *서비스*(*service*) obtains 1 point in "*야후코리아 (www.yahoo.co.kr)가 새로운 서비스를 시작한다.* (*Yahoo Korea (www.yahoo.co.kr) starts a new service.*)".
- Grammatical Role: the grammatical relations of the subcategorized functions of the main verb in a sentence. The indexing engine gives 4 points to a topic word, 3 points to a subject, 2 points to an object and 1 point to the rests. The grammatical roles can be decided by case markers like *은/는*(*un/nun*), *이/가*(*i/ga*) and *을/를*(*ul/lul*) since Korean is a language with well-developed morphemic markers. For example, *야후 코리아*(*Yahoo Korea*) obtains 3 points because it is a subject, and *서비스*(*service*) obtains 2 point because it is an object in the above sample sentence.
- Lexical Chain: the re-occurring words in adjacent sentences. The indexing engine gives 2 points to each word that forms lexical chains and gives 1 point to others. For example, if the next sentence of the above sample sentence is "*그 서비스를 사용하는 고객은 6 메가의 무료 이메일 공간을 사용할 수 있다.* (*The members of the service can use the free storages of 6 mega-bytes for email.*)", *서비스* (*service*) obtains 2 points.
- Distance: the distance between a sentence including a target content word and a sentence including an answer candidate. The indexing engine gives 2 points to each content word in the sentence including the answer candidate.

The engine gives 1 point to others. For example, *야후코리아*(*Yahoo Korea*) and *서비스*(*service*) in the above sample sentence obtain 2 points respectively because the content words are in the sentence including the answer candidate, *www.yahoo.co.kr*.

- Apposition: the IS-A relation between a content word and an answer candidate. The indexing engine extracts appositive terms by using syntactic information such as Explicit IS-A relation, Pre-modification and Post-modification. For example, *야후코리아*(*Yahoo Korea*) is Pre-modification relation with *www.yahoo.co.kr* in the above sample sentence. The indexing engine gives 2 points to each appositive word and gives 1 point to others.

The indexing engine adds up the scores of the 5 features, as shown in Equation 1.

$$ts_i = \frac{A \cdot f_{i1} + B \cdot f_{i2} + C \cdot f_{i3} + D \cdot f_{i4} + E \cdot f_{i5}}{A + B + C + D + E} \qquad (1)$$

ts_i is the term score of the *i*th term, and f_{ij} is the score of the *j*th feature in the *i*th term. *A*, *B*, *C*, *D* and *E* are weighting factors that rank 5 features according to preference. The indexing engine uses the following ranking order: $E > C > B > A > D$. The weighted term scores are normalized, as shown in Equation 2.

$$\begin{cases} \left(0.5 + 0.5\frac{ts_{ij}}{Max\_ts_j}\right)\frac{\log(N/n)}{\log(N)} & ts_{ij} > 0 \\ 0 & ts_{ij} = 0 \end{cases} \qquad (2)$$

Equation 2 is similar to TF·IDF equation (Fox, 1983). In Equation 2, ts_{ij} is the term score of the *i*th term in the context window that is relevant to the *j*th answer candidate. Max_ts_j is the maximum value among term scores in the context window that is relevant to the *j*th answer candidate. *n* is the number of answer candidates that are affected by the *i*th term. *N* is the number of answer candidates of the same semantic category. The indexing engine saves the normalized term scores with the position information of the relevant answer candidate in the DB. The position information includes a document number and the distance between the beginning of the document and the answer candidate. As a result, the indexing engine creates 14 DB's that correspond to the 14 semantic categories. We call them answer DB's.

3.2 Lexico-syntactic Query processing

In the query processing stage, the searching engine takes a user's question and converts it into a suitable form, using a semantic dictionary, called a query dictionary. The query dictionary contains the semantic markers of words. Query words are converted into semantic markers before pattern matching. For example, the query "*야후코리아의 사장은 누구인가요?* (*Who is the CEO of Yahoo Korea?*)" is translated into "*야후코리아 j %사람 j %누구 jp ef sf (%who auxiliary-verb %person preposition Yahoo Korea symbol*)". In the example, *%사람* (*%person*) and *%누구*(*%who*) are the semantic markers. The content words out of the query dictionary keep their lexical forms. The functional words (e.g. auxiliary verb, preposition) are converted into POS's. After conversion, the searching engine matches the converted query against one of 88 lexico-syntactic patterns, and classifies the query into the one of 14 semantic categories. When two or more patterns match the query, the searching engine returns the first matched category.

```
%사람 (xsn)* (j)?%누구.* $
(%person (xsn)* (j)? %who .* $)

%사람 (xsn)* (j)? %이름 (j) (%무엇)? .* $
(%person (xsn)* (j)? %name (j) (%what)? .* $)

%사람 (xsn)* (j)? (%이름)? %요구 .* $
(%person (xsn)* (j)? (%name)? %want_to_know .* $)

%어느 %사람 .* $
(%which %person .* $)
```

Figure 4. Lexico-syntactic patterns

Figure 4 shows some lexico-syntactic patterns for *person* category. The above sample query matches the first pattern in Figure 4.

After classifying the query into a semantic category, the searching engine calculates the term scores of the content words in the query. As shown in Rule 1, the term scores are computed by some heuristic rules, and the range of the term scores is between 0 and 1. Using the heuristic rules, the searching engine gives high scores to content words that focus a user's intention. For example, when a user inputs the query "*야후는 몇 년도에 설립되었나?* (*In what year is Yahoo founded?*)", he/she wants to know only the year, rather than the organizer or the URL of Yahoo. So, the QA searching engine

gives a higher score to 년도(*year*) than to 야후 (*Yahoo*) in contrast to the ordinary IR searching engine.

1. The last content word in a sentence receives a high score. For example, 사장 (*CEO*) in "야후 사장은? (*The CEO of Yahoo?*)" receives a high score.
2. The next content words of specific interrogatives such as 어느 (*which*), 무슨 (*what*) receive high scores. For example, 산(*mountain*) in "어느 산이 가장 높은가? (*Which mountain is the highest?*)" receives a high score.
3. The next content words of specific prepositions like 관한 (about) receive low scores, and the previous content words receive high scores. For example, the score of 기사 (*article*) in "중국에 관한 기사 (*the article about China*)" is lower than that of 중국(*China*).

Rule 1. Heuristic rules for scoring query terms

3.3 Answer scoring and ranking

The searching engine calculates the similarities between query and answer candidates, and ranks the answer candidates according to the similarities. To check the similarities, the searching engine uses the AND operation of a well-known p-Norm model (Salton et al., 1983), as shown in Equation 3.

$$Sim(A, Q_{and}) = 1 - \sqrt[p]{\frac{q_1^p(1-a_1)^p + q_2^p(1-a_2)^p + \cdots + q_i^p(1-a_i)^p}{q_1^p + q_2^p + \cdots + q_i^p}} \quad (3)$$

In Equation 3, A is an answer candidate, and a_i is the ith term score in the context window of the answer candidate. a_i is stored in the answer DB. q_i is the ith term score in the query. p is the P-value in the p-Norm model.

It takes a relatively short time for answer scoring and ranking phase because the indexing engine has already calculated the scores of the terms that affect answer candidates. In other words, the searching engine simply adds up the weights of co-occurring terms, as shown in Equation 3. Then, the engine ranks answer candidates according to the similarities. The method for answer scoring is similar to the method for document scoring of traditional IR engines. However, MAYA is different in that it indexes, retrieves, and ranks answer candidates, but not documents.

We can easily combine MAYA with a traditional IR system because MAYA has been designed by a separate component that interfaces with the IR system. We implemented an IR system that is based on TF·IDF weight and p-Norm model (Lee et al., 1999).

To improve the precision rate of the IR system, we combine MAYA with the IR system. The total system merges the outputs of MAYA with the outputs of the IR system. MAYA can produce multiple similarity values per document if two or more answer candidates are within a document. However, the IR system produces a similarity value per document. Therefore, the total system adds up the similarity value of the IR system and the maximum similarity value of MAYA, as shown in Equation 4.

$$Sim(D,Q) = \frac{\alpha \cdot IRsim(D,Q) + \beta \cdot QAsim_d(A_i,Q)}{\alpha + \beta} \quad (4)$$

In Equation 4, $QAsim_d(A_i,Q)$ is the similarity value between query Q and the ith answer candidate A_i in document d. $IRsim(D,Q)$ is the similarity value between query Q and document D. α and β are weighting factors. We set α and β to 0.3 and 0.7.

The total system ranks the retrieved documents by using the combined similarity values, and shows the sentences including answer candidates in the documents.

4 Evaluation

4.1 The experiment data

In order to experiment on MAYA, we collected 14,321 documents (65,752 kilobytes) from two web sites: *korea.internet.com* (6,452 documents) and *www.sogang.ac.kr* (7,869 documents). The former gives the members on-line articles on Information Technology (IT). The latter is a homepage of Sogang University. The indexing engine created the 14 answer DBs (14 semantic categories).

For the test data, we collected 50 pairs of question-answers from 10 graduate students. Table 1 shows the 14 semantic categories and the numbers of the collected question-answers in each category. As shown in Table 1, we found 2 question-answers out of the 14 semantic categories. They are not closed-class question-

answers but explanation-seeking question-answers like "*Question: How can I search on-line Loyola library for any books? Answer: Connect your computer to http://loyola 1.sogang.ac.kr*".

Category	person	country	address	organization
# of QAs	9	3	3	9
Category	telephone	email	URL	people num.
# of QAs	3	5	4	0
Category	phy. num.	abs. num.	rate	price
# of QAs	1	1	0	4
Category	date	time	out of cat.	total
# of QAs	5	1	2	50

Table 1. The number of the collected question-answers in each category

We use two sorts of evaluation schemes. To experiment on MAYA, we compute the performance score as the Reciprocal Answer Rank (RAR) of the first correct answer given by each question. To compute the overall performance, we use the Mean Reciprocal Answer Rank (MRAR), as shown in Equation 5 (Voorhees and Tice, 1999).

$$MRAR = 1/n\left(\sum_i 1/rank_i\right) \quad (5)$$

With respect to the total system that combines MAYA with the IR system, we use the Reciprocal Document Rank (RDR) and the Mean Reciprocal Document Rank (MRDR). RDR means the reciprocal rank of the first document including the correct answers given by each question.

4.2 Analysis of experiment results

The performance of MAYA is shown in Table 2. We obtained the correct answers for 33 questions out of 50 in Top 1.

Rank	Top 1	Top 2	Top 3	Top 4
# of answers	33	4	3	2
Rank	Top 5	Top 6~	Failure	Total (MRAR)
# of answers	1	2	5	50 (0.80)

Table 2. The performance of the QA system

Table 3 shows the performance of the total system. As shown in Table 3, the total system significantly improves the document retrieval performance of underlying IR system about the closed-class questions.

The average retrieval time of the IR system is 0.022 second per query. The total system is 0.029 second per query. The difference of the retrieval times between the IR system and the total system is not so big, which means that the retrieval speed of QA-only-system is fast enough to be negligible. The IR system shows some sentences including query terms to a user. However, the total system shows the sentences including answer candidates to a user. This function helps the user get out of the trouble that the user might experience when he/she looks through the whole document in order to find the answer phrase.

Rank	Top 1	Top 2	Top 3	Top 4
# of answers 1	22	8	5	2
# of answers 2	36	5	2	1
Rank	Top 5	Top 6~	Failure	Total (MRDR)
# of answers 1	3	10	0	50 (0.54)
# of answers 2	2	4	0	50 (0.76)

# of answers 1: the number of answers which are ranked at top n by using the IR system
# of answers 2: the number of answers which are ranked at top n by using the total system

Table 3. The performance of the total system

MAYA could not extract the correct answers to certain questions in this experiment. The failure cases are the following, and all of them can be easily solved by extending the resources and pattern rules:

- The lexico-syntactic parser failed to classify users' queries into the predefined semantic categories. We think that most of these failure queries can be dealt with by supplementing additional lexico-syntactic grammars.
- The NE recognizer failed to extract answer candidates. To resolve this problem, we should supplement the entries in PLO dictionary, the entries in the unit dictionary, and regular expressions. We also should endeavor to improve the precision of the NE recognizer.

5 Conclusion

We presented a fast and high-precision Korean QA system using a predictive answer indexer. The predictive answer indexer extracts answer candidates and terms surrounding the candidates in indexing time. Then, it stores each candidate with the surrounding terms that have specific scores in answer DB's. On the retrieval time, the QA system just calculates the similarities between a user's query and the answer candidates. Therefore, it can minimize the retrieval time and enhance the precision. Our system can easily converted into other domains because it is based on shallow NLP and IR techniques such as POS tagging, NE recognizing, pattern matching and term weighting with TF·IDF. The experimental results show that the QA system can improve the document retrieval precision for closed-class questions after the insignificant loss of retrieval time if it is combined with a traditional IR system. In the future, we pursue to concentrate on resolving the semantic ambiguity when a user's query matches two or more lexico-syntactic patterns. Also, we are working on an automatic and dynamic way of extending the semantic categories into which the users' queries can be more flexibly categorized.

References

AAAI Fall Symposium on Question Answering. 1999.

Berri, J., Molla, D., and Hess, M. 1998. Extraction automatique de réponses: implémentations du systéme ExtrAns. In *Proceedings of the fifth conference TALN 1998*, pp. 10-12.

Ferret, O., Grau, B., Illouz, G., and Jacquemin C. 1999. QALC – the Question- Answering program of the Language and Cognition group at LIMSI-CNRS. In *Proceedings of The Eighth Text REtrieval Conference*(*TREC-8), http://trec.nist. gov/pubs/trec8/t8_proceedings.html*.

Fox, E.A. 1983. *Extending the Boolean and Vector Space Models of Information Retrieval with P-norm Queries and Multiple Concept Types*, Ph.D. Thesis, CS, Cornell University.

Hull, D.A. 1999. Xerox TREC-8 Question Answering Track Report. In *Proceedings of The Eighth Text REtrieval Conference(TREC-8), http://trec.nist.gov/pubs/trec8/t8_proceedings.html*.

Kupiec, J. 1993. Murax: A Robust Linguistic Approach for Question Answering Using an On-line Encyclopedia. In *Proceedings of SIGIR'93*.

Lee, G., Park, M., and Won, H. 1999. Using syntactic information in handling natural language queries for extended boolean retrieval model. In *Proceedings of the 4th international workshop on information retrieval with Asian languages (IRAL99)*, pp. 63-70.

Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Goodrum, R., Gîrju, R., and Rus, V. 1999. LASSO: A Tool for Surfing the Answer Net. In *Proceedings of The Eighth Text REtrieval Conference (TREC-8), http://trec.ni st.gov/pubs /trec8/t8_proceedings.html*.

Prager, J., Brown, E., Coden A., and Radev D. 2000. Question-Answering by Predictive Annotation. In *Proceedings of SIGIR 2000*, pp. 184-191.

Prager, J., Radev, D., Brown, E., and Coden, A. 1999. The Use of Predictive Annotation for Question Answering in TREC8. In *Proceedings of The Eighth Text REtrieval Conference (TREC-8), http://trec.nist.gov/pubs/trec8/t8_proceedings.html*.

Salton, G., Fox, E.A., and Wu, H. 1983. Extended Boolean Information Retrieval, *Communication of the ACM*, 26(12):1022-1036.

Salton, G., and McGill, M. 1983. *Introduction to Modern Information Retrieval (Computer Series)*, New York:McGraw-Hill.

Salton, G. 1989. *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer*. Reading, MA:Addison-Wesley.

TREC (Text REtrieval Conference) Overview, http://trec.nist.gov/overview.html.

Sparck-Jones, K. 1999. What is the role NLP in Text Retrieval?. *Natural Language Information Retrieval*, Kluwer Academic Publishers. T.Strzalkowski (ed), pp.1-24.

Srihari, R., and Li, W. 1999. Information Extraction Supported Question Answering. In *Proceedings of The Eighth Text REtrieval Conference (TREC-8), http://trec.nist.gov/pubs/trec8/t8_proceedings.html*.

Vicedo, J. L., and Ferrández, A. 2000. Importance of Pronominal Anaphora resolution in Question Answering systems. In *Proceeding of ACL 2000*, pp. 555-562.

Voorhees, E., and Tice, D. M. 1999. The TREC-8 Question Answering Track Evaluation. In *Proceedings of The Eighth Text REtrieval Conference (TREC-8), http://trec.nist.gov/pubs /trec8/t8_proceedings.html*.