A Reliable Indexing Method for a Practical QA System

Harksoo Kim
Diquest Inc.
Sindo B/D, 1604-22, Seocho-dong
Seocho-gu, Seoul, Korea, 137-070
hskim@diquest.com

Jungyun Seo
Department of Computer Science
Sogang University, 1 Sinsu-dong,
Mapo-gu, Seoul, Korea, 121-742
seojy@ccs.sogang.ac.kr

Abstract

We propose a fast and reliable Question-answering (QA) system in Korean, which uses a predictive answer indexer based on 2-pass scoring method. The indexing process is as follows. The predictive answer indexer first extracts all answer candidates in a document. Then, using 2-pass scoring method, 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 saves response time and enhances the precision.

Introduction

Traditional Information Retrieval (IR) focuses on searching and ranking a list of documents in response to a user's question. However, in many cases, a user has a specific question and want for IR systems to return the answer itself rather than a list of documents (Voorhees and Tice (2000)). To satisfy this need, the concept of Question Answering (QA) comes up, and a lot of researches have been carried out, as shown in the proceedings of AAAI (AAAI (n.d.)) and TREC (Text REtrieval Conference) (TREC (n.d.)). A QA system searches a large collection of texts, and filters out inadequate phrases or sentences within the texts. Owing to the filtering process, a user can promptly approach to his/her answer phrases without troublesome tasks. Unfortunately, most of the previous researches have passed over the following problems that occurs in real fields like World Wide Web (WWW):

- Users want to find answers as soon as possible. If a QA system does not respond to their questions within a few seconds, they will keep a suspicious eye on usefulness of the system.
- Users express their intentions by using various syntactic forms. The fact makes it difficult that a QA system performs well at any domains. Ultimately, the QA system cannot be easily converted into any domains.
- A QA system cannot correctly respond to all of the users' questions. It can answer the questions that are included in the predefined categories such as *person*, *date*, and *time*.

To solve the problems, we propose a practical QA system using a predictive answer indexer in Korean - MAYA (MAke Your Answer). MAYA focuses on resolving the practical problems such as real-time response and domain portability. 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. Users can promptly obtain answer phrases on retrieval time because MAYA indexes answer candidates in advance.

This paper is organized as follows. First, we review the previous works of the QA systems. Second, we present our system, and describe the applied NLP techniques. Third, we analyze the result of our experiments. Finally, we draw conclusions.

1 Previous works

The current QA approaches can be classified into two groups; text-snippet extraction methods and noun-phrase extraction methods (also called closed-class QA) (Vicedo and Ferrández (2000)). The text-snippet extraction methods 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 methods have been the most commonly used by participants in last TREC QA Track (Moldovan et al. (1999); Prager, Radev, Brown and Coden (1999)). The noun-phrase extraction methods 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 a definite answer typified by a noun phrase rather than a procedural answer.

ExtrAns (Berri, Molla and Hess (1998)) is a representative QA system using the text-snippet extraction method. 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)). FALCON (Harabagiu et al. (2000)) is another text-snippet system. The system returns answer phrases with high precision because it integrates different forms of syntactic, semantic and pragmatic knowledge for the goal of archiving better performance. The answer engine of FALCON handles question reformulations of previously posed questions, finds the expected answer type from a large hierarchy that incorporates the WordNet (Miller (1990)), and extracts answers after performing unifications on the semantic forms of the question and its answer candidates. Although FALCON archives good performance, the system is not appropriate for a practical QA system because it is difficult to construct domain-specific knowledge like a semantic net.

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 TREC use a shallow linguistic knowledge and start from similar approaches as used in MURAX (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. To overcome the week point, GuruQA system (Prager, Brown and Coden (2000)), one of text-snippet systems, uses a method for indexing answer candidates in advance (so-called Predictive Annotation). Predictive Annotation identifies answer candidates in a text, annotates them accordingly, and indexes them. Although the GuruQA system quickly replies to users' queries and has good performance, the system passed over useful information out of a document boundary. In other words, the system restricts the size of a context window containing an answer candidate from a sentence to a whole document, and calculates a similarity between the keywords in a query and the keywords in the window. The system does not consider any information out of the window at all.

2 Approach of MAYA

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. As shown in Figure 1, it consists of two engines; an indexing engine and a searching engine.

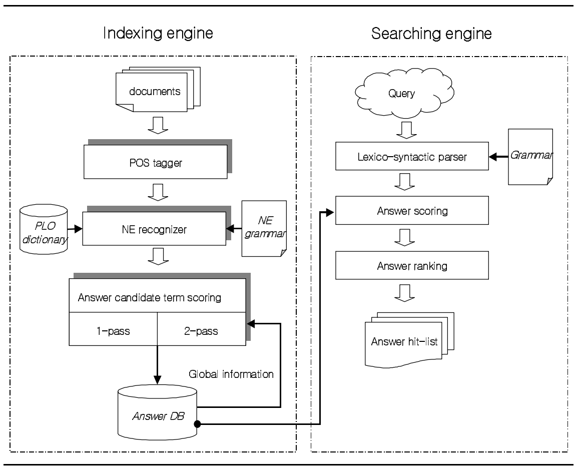


Figure 1. A basic architecture of MAYA

2.1 Predictive answer indexing

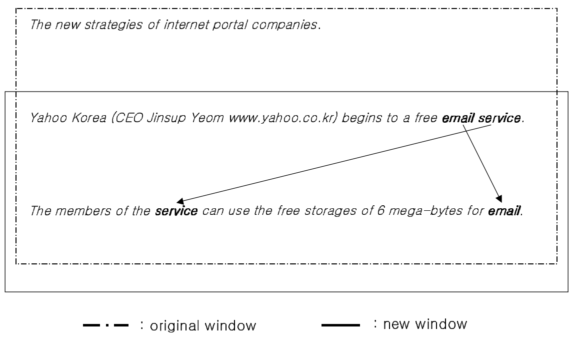
The answer indexing phase can be separated in 2 stages; answer-finding and term-scoring. For answer-finding, we classify users' asking points into 105 semantic categories. As shown in Table 1, The 105 semantic categories consist of 2 layers; the first layer and the second layer. The semantic categories in the first layer have broader meanings than those in the second layer. To define the 105 categories, we referred to the categories of QA systems participating in TREC and analyzed users' query logs that are collected by a commercial IR system (DiQuest.com (n.d.)).

Table 1. A part of 105 semantic categories

The first layer	The second layer		
animal	bird	fish	mammal
	person	reptile	
location	address	building	city
	continent	country	state
	town		
date	day	month	season
	weekday	year	
time	hour	minute	second
organization	company	department	family
	group	laboratory	school
	team		

To extract answer candidates belonging to each category from documents, the indexing engine uses a POS tagger and a NE recognizer. The NE recognizer consists of a named entity dictionary (so-called PLO dictionary) and a pattern matcher. The PLO dictionary contains not only the names of people, countries, cities, and organizations, but it also contains a lot of units such as the unit of the length (e.g. *cm*, *m*, *km*) and the units of weight (e.g. *mg*, *g*, *kg*). After looking up the dictionary, 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, "*Yahoo Korea (CEO Jinsup Yeom www.yahoo.co.kr) expanded the size of the storage for free email service to 6 mega-bytes.*". *Yahoo Korea* belongs to *company*, and *Jinsup Yeom* is *person*. *www.yahoo.co.kr* means *URL*, and *6 mega-bytes* is *size*. The 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 *URL*. The patterns are described as regular expressions.

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 anaphors or lexical chains. 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 2 shows an example in which the window size is adjusted.


The new strategies of internet portal companies.

Yahoo Korea (CEO Jinsup Yeom www.yahoo.co.kr) begins to a free ***email service****.*

The members of the ***service*** *can use the free storages of 6 mega-bytes for* ***email****.*

Figure 2. An example with the adjusted window size

After setting the context window, the indexing engine assigns scores to the content words in the window by using a 2-pass scoring method. In the first pass, the indexing engine calculates *local scores* of the content words. The scores indicate the magnitude of influences that each content word causes to answer candidates in a document. For example, when *www.yahoo.co.kr* is an answer candidate in the sentence, "*Yahoo Korea (www.yahoo.co.kr) starts a new service.*", *Yahoo Korea* has the higher score than *service* since it has much more strong clue to *www.yahoo.co.kr*. We call the score a *local score* because the score is obtained from information between two adjacent words in a document. The indexing engine assigns *local scores* to content words according to 2 scoring features described below.

- Term frequency: the frequency of each content word in a context window. The indexing engine give high scores to content words that frequently occurs with answer candidates For example, *email* receives a higher score than *members* in Figure 2.

- Distance: the distance between an answer candidate and a target content word. The indexing engine gives high scores to content words that are near to answer candidates. For example, when *Jinsup Yeom* is an answer candidate in Figure 2, *CEO* obtains a higher score than *service*.

The indexing engine does not use high-level information like definition characteristics (IS-A relation between words in a sentence) and grammatical roles because it is difficult for the indexing engine to correctly extract the high-level information from documents in real fields. In other words, most of the web documents are described in a user's free style with additional tags and includes a lot of images and tables. The fact makes it more difficult for the indexing engine to detect sentence boundaries and to extract topic words from sentences. Therefore, the indexing engine uses law-level information like the term frequencies and the distances after considering the cost for the additional analysis and indexing time.

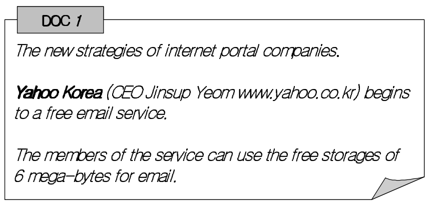
The indexing engine calculates *local scores* by two steps. It first calculates the distance weight between an answer candidate and a target content word, as shown in Equation 1.

$$distw_{d,k}(a_i, w_j) = \frac{c}{\log(dist(i, j)) + c} \qquad (1)$$

In Equation 1, $distw_{d,k}(a_i, w_j)$ is the distance weight of the content word *w* that is located at the *j*th position in the *k*th context window of a document *d*. $dist(i, j)$ is the distance between the answer candidate a_i, which is located at the *i*th position, and the content word w_j, which is located at the *j*th position. *c* is a constant value, and we set *c* to 1 on experiment. The indexing engine then adds up the distance weights of content words with an identical lexical form in each context window, as shown in Equation 2.

$$\begin{aligned} LS^n_{d,k}(a_i, w_{pos(n)}) &= distw_{d,k}(a_i, w_{pos(n)}) + \\ &\left(1 - distw_{d,k}(a_i, w_{pos(n)})\right) \times LS^{n-1}_{d,k}(a_i, w_{pos(n-1)}), \qquad (2) \\ &where\ LS^0_{d,k}(a_i, w_{pos(0)}) = 0. \end{aligned}$$

Equation 2 is described as a well-known dynamic programming method. According to Equation 2, the more frequent content words are, the higher scores the content words receive. In Equation 2, $LS^n_{d,k}(a_i, w_{pos(n)})$ is the *local score* of the *n*th content word *w* when *n* identical content words exist in the *k*th context window of a document *d*, and *pos(n)* is the position of the *n*th content word. After recursively solving Equation 2, the indexing engine receives a *local score*, $LS_{d,k}(a_i, w)$, between the *i*th answer candidate and the content word *w* in the *k*th context window. Figure 3 shows the calculation process of *local scores*. After calculating the *local scores*, the indexing engine saves the *local scores* with the position information of the relevant answer candidate in the answer DB.



DOC *1*

The new strategies of internet portal companies.

Yahoo Korea *(CEO Jinsup Yeom www.yahoo.co.kr) begins to a free email service.*

The members of the service can use the free storages of 6 mega-bytes for email.

Answer candidate	Process
Yahoo Korea	Measure the distances between *Yahoo Korea* and each *service* that is located in the two adjacent sentences. dist(1, 7) = 6, dist(1, 9) = 8 Calculate each distance weight. *distw*(*Yahoo Korea$_1$*, *service$_7$*) = 1/(log(6)+1)=0.358 *distw*(*Yahoo Korea$_1$*, *service$_9$*) = 1/(log(8)+1)=0.325 Add up the distance weights. *LS*(*Yahoo Korea$_1$*, *service*) = 0.358+(1.0-0.358)*0.325=0.567

Figure 3. An example of the local scores

The second pass is divided into three steps; construction of *pseudo-documents*, calculation of *global scores*, and summation of *global scores* and *local scores*. In the first step, the indexing engine constructs *pseudo-documents*. A *pseudo-document* is a virtual document that consists of content words occurring with an answer candidate in some documents. The *pseudo-document* is named after the answer candidate. Figure 4 shows an example of the *pseudo-documents*.

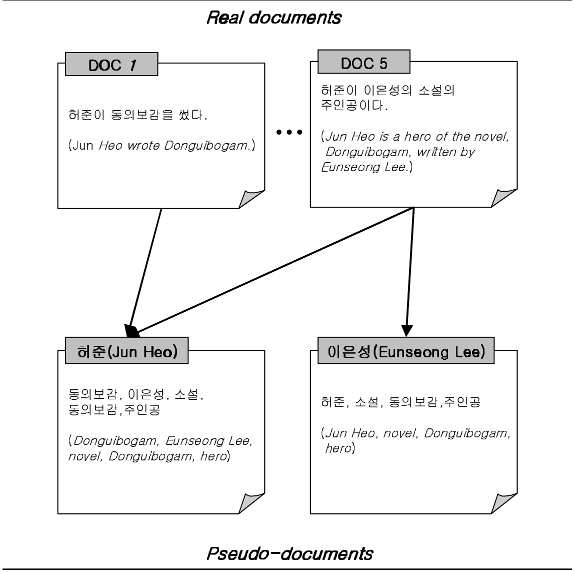
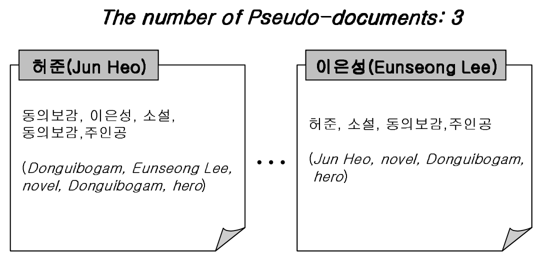

Real documents
DOC 1
허준이 동의보감을 썼다.
(Jun Heo wrote Donguibogam.)
DOC 5
허준이 이은성의 소설의 주인공이다.
(Jun Heo is a hero of the novel, Donguibogam, written by Eunseong Lee.)
허준(Jun Heo)
동의보감, 이은성, 소설, 동의보감,주인공
(Donguibogam, Eunseong Lee, novel, Donguibogam, hero)
이은성(Eunseong Lee)
허준, 소설, 동의보감,주인공
(Jun Heo, novel, Donguibogam, hero)
Pseudo-documents

Figure 4. An example of the pseudo-documents

In the next step, the indexing engine calculates *global scores* of each answer candidate, as shown in Equation (3). The *global score* mean how much the answer candidate is associated with each term that occurs in several documents.

$$GS(pseudo\_d_a, w) = \begin{cases} \left(0.5+0.5\dfrac{tf_w}{Max\_tf}\right)\dfrac{\log(N/n)}{\log(N)}, & if \quad tf_w > 0 \\ 0, & if \quad tf_w = 0 \end{cases} \tag{3}$$

Equation 3 is similar to a well-known TF·IDF equation (Fox (1983)). However, the equation is different when it comes to the concept of a document. We assume that there is no difference between a *pseudo-document* and a real document. Therefore, the TF component, $(0.5+0.5\cdot(tf_w / Max\_tf))$ in Equation 3, means the normalized frequency of the content word *w* in the *pseudo-document* $pseudo\_d_a$ that is named after the answer candidate *a*. The IDF component, $\log(N/n)/\log(N)$, means the normalized reciprocal frequency of the *pseudo-documents* including the content word *w*. The value of TF·IDF, $GS(pseudo\_d_a, w)$, means the *global score* between the answer candidate *a* and the content word *w*. In detail, tf_w is the term frequency of the content word *w* in $pseudo\_d_a$. *Max_tf* is the maximum value among the frequencies of content words in $pseudo\_d_a$. *n* is the number of the *pseudo-documents* that include the content word *w*. *N* is the total number of the *pseudo-documents*. Figure 5 shows a calculation process of the *global scores*.


The number of Pseudo-documents: 3
허준(Jun Heo)
동의보감, 이은성, 소설, 동의보감,주인공
(Donguibogam, Eunseong Lee, novel, Donguibogam, hero)
이은성(Eunseong Lee)
허준, 소설, 동의보감,주인공
(Jun Heo, novel, Donguibogam, hero)

허준 (*Jun Heo*)

GS(*Jun Heo,Donguibogam*) =
(0.5+0.5*(2/2))*(log(3/2)/log(3)) = 0.369

GS(*Jun Heo,Eunseong Lee*) =
(0.5+0.5*(1/2))*(log(3/2)/log(3)) = 0.277

GS(*Jun Heo,novel*) =
(0.5+0.5*(1/2))*(log(3/2)/log(3)) = 0.277

GS(*Jun Heo,hero*) =
(0.5+0.5*(1/2))*(log(3/2)/log(3)) = 0.277

Figure 5. An example of the global scores

In the last step, the indexing engine adds up the *global scores* and the *local scores*, as shown in Equation (4).

$$S_{d,k}(a_i, w) = \frac{\alpha \cdot LS_{d,k}(a_i, w) + \beta \cdot GS(pseudo\_d_{a_i}, w)}{\alpha + \beta} \tag{4}$$

In Equation 4, $LS_{d,k}(a_i, w)$ is the *local score* between the answer candidate a_i and the content word *w* in the *k*th content window of the document *d*, and $GS(pseudo\_d_{a_i}, w)$ is the *global score*. α and β are weighting factors. After summing up two scores, the indexing engine updates the answer DB with the scores.

2.2 Lexico-syntactic query processing

For identifying users' asking points, the searching engine takes a user's query and converts it into a suitable form using the PLO dictionary. The PLO dictionary contains the semantic markers of words. Query words are converted into semantic markers before pattern matching. For example, the query "*Who is the CEO of Yahoo Korea?*" is translated into "*%who auxiliary-verb %person preposition Yahoo Korea symbol*". In the example, *%person* and *%who* are the semantic markers. The content words out of the PLO 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 predefined lexico-syntactic patterns, and classifies the query into the one of the 105 semantic categories. When two or more patterns match the query, the searching engine returns the first matched category. Table 2 shows some lexico-syntactic patterns. The above sample query matches the first pattern in Table 2.

Table 2. Lexico-syntactic patterns

Semantic category	Lexico-syntactic patterns
person	*%who (j|ef)?* *(%person|@person) j? (sf)* $* *(%person|@person) j? %ident j? (sf)* $* *(%person|@person) j? (%about)? @req* *(%person|@person) j? (%ident)? @req* *(%person|@person) jp ef (sf)* $* *%which (%person|@person)*
tel_num	*(%tel_num|@tel_num) (%num)? j? (sf)*$* (%tel_num|@tel_num) (%num)? j? %what *(%tel_num|@tel_num) j? (%about)? @req* *(%tel_num|@tel_num) j? (%what_num)*

2.3 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, Fox and Wu (1983)), as shown in Equation 5.

$$Sim(A, Q_{and}) = 1 - \sqrt[p]{\frac{q_1^p(1-at_1)^p + q_2^p(1-at_2)^p + \cdots + q_i^p(1-at_i)^p}{q_1^p + q_2^p + \cdots + q_i^p}} \qquad (5)$$

In Equation 5, A is an answer candidate, and at_i is the ith term score in the context window of the answer candidate. q_i is the ith term score in the query. p is the P-value in the p-Norm model.

MAYA consumes 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 5. 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.

3 Evaluation

3.1 The Experiment data

To experiment on MAYA, we use two sorts of document collections. One is a collection of documents that are collected from two web sites; *korea.internet.com* and *www.sogang.ac.kr*. The former gives the members on-line articles on Information Technology (IT). The latter is a homepage of Sogang University. We call the collection WEBTEC (WEB TEst Collection). The other is KorQATeC 1.0 (Korean Test Collection for evaluation of QA system) (Lee, Kim and Choi (2000)). WEBTEC consists of 22,448 documents (110,004 kilobytes), and KorQATeC 1.0 consists of 207,067 balanced documents (368,768 kilobytes). WEBTEC and KorQATeC 1.0 each include 50 pairs of question-answers (QAs).

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 6 (TREC (n.d.); Voorhees and Tice (1999)).

$$MRAR = 1/n\left(\sum_i 1/rank_i\right) \qquad (6)$$

In Equation 6, $rank_i$ is the rank of the first correct answer given by the ith question. n is the number of questions.

3.2 The analysis of experiment results

For ranking answer candidates, MAYA uses the weighted sums of *global scores* and *local scores*, as shown in Equation 4. To set the weighting factors, we evaluated performances of MAYA according to the values of the weighting factors. Table 3 shows overall MRAR as the values of the weighting factors are changed. In Table 3, the boldface MRARs are the highest scores in each test bed. We set α and β to 0.1 and 0.9 on the basis of the experiment.

Table 3. The performances of MAYA according to the values of the weighting factors

α	β	WEBTEC	KorQATeC	TOTAL
1.0	0.0	0.354	0.506	0.435
0.9	0.1	0.341	0.506	0.430
0.8	0.2	0.350	0.520	0.444
0.7	0.3	0.365	0.524	0.452
0.6	0.4	0.379	0.526	0.462
0.5	0.5	0.388	0.515	0.466
0.4	0.6	0.388	0.516	0.471
0.3	0.7	0.385	0.519	0.461
0.2	0.8	**0.405**	0.524	0.471
0.1	0.9	0.395	**0.540**	**0.473**
0.0	1.0	0.349	0.475	0.438

To evaluate the performance of MAYA, we compared MAYA with Lee2000 (Lee, Kim and Choi (2000)) and Kim2001 (Kim, Kim, Lee and Seo (2000)) in KorQATeC 1.0 because we could not obtain any experimental results on Lee2000 in WEBTEC. As shown in Table 4, the performance of MAYA is higher than those of the other systems. The fact means that the scoring features of MAYA are useful. In Table 4, Lee2000 (50-byte) returns 50-byte span of phrases that include answer candidates, and the others return answer candidates in themselves. MRAR-1 is MRAR except questions for which the QA system fails in finding correct answers.

Table 4. The performances of the QA systems in KorQATeC 1.0

	Lee2000 (object)	Lee2000 (50-byte)	Kim2001 (object)	MAYA (object)
MRAR	0.322	0.456	0.485	0.540
MRAR-1	0.322	0.456	0.539	0.600

MAYA could not extract correct answers for 5 questions. The failure cases are the following:

- The query classifier failed to identify users' asking points. 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 the PLO dictionary and regular expressions. We also should endeavor to improve the precision of the NE recognizer.

Table 5. The difference of response times

	Response time per query (seconds)	Indexing time per mega byte (seconds)
IR system	0.026	2.830
MAYA	0.048	19.120
Incomplete -MAYA	5.300	2.830

As shown in Table 5, the average retrieval time of the IR system (Lee, Park and Won (1999)) is 0.026 second per query on a PC server with dual Intel Pentium III. MAYA consumes 0.048 second per query. The difference of the retrieval times between the IR system and MAYA is not so big, which means that the retrieval speed of MAYA is fast enough to be negligible. Table 5 also shows the difference of the response times between MAYA and a QA system without a predictive answer indexer. We call the QA system without an answer indexer Incomplete-MAYA. Incomplete-MAYA finds and ranks answer candidates on retrieval time. Hence, it does not need additive indexing time except indexing time for the underlying IR system. In the experiment on the response time, we made Incomplete-MAYA process answer candidates just in top 30 documents that are retrieved by the underlying IR system. If Incomplete-MAYA finds and ranks answer candidates in the whole retrieved documents, it will take longer response time than the response time in Table 5. As shown in Table 5, the response time of MAYA is about 110 times faster than that of Incomplete-MAYA. Although MAYA consumes 19.120 seconds per mega byte for creating the answer DB, we conclude that MAYA is more efficient because most of the users are impatient for a system to show answers within a few milliseconds.

4 Conclusion

We presented a fast and high-precision QA system using a predictive answer indexer in

Korean. The predictive answer indexer extracts answer candidates and terms adjacent to the candidates on the indexing time. Then, using the 2-pass scoring method, the indexer stores each candidate with the adjacent terms that have specific scores in the answer DB. On the retrieval time, the QA system just calculates the similarities between a user's query and the answer candidates. Therefore, the QA system minimizes the retrieval time and enhances the precision. Moreover, 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.

References

AAAI Fall Symposium on Question Answering (n.d.) Retrieved April 22, 2002, from http://www.aaai.org/Press/Reports/Symposia/Fall/fs-99-02.html

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.

DiQuest.com (n.d.) http://www.diquest.com

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.

Harabagiu S., Moldovan D., Pasca M., Mihalcea R., Surdeanu M., Bunescu R., Gîrju R., Rus V. and Morarescu P. (2000) *FALCON: Boosting Knowledge for Answer Engines*. In "Proceedings of the Eighth Text REtrieval Conference (TREC-9)", from http://trec.nist.gov/pubs/trec9/t9_proceedings.html

Kim H., Kim K., Lee G. G. and Seo J. (2001) *MAYA: A Fast Question-answering System Based On A Predictive Answer Indexer*. In "Proceedings of the ACL Workshop Open-Domain Question Answering", pp. 9-16.

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)", Academia Sinica, Taipei, pp. 63-70.

Lee K., Kim J. and Choi, K. (2000) *Answer Extraction based on Named Entity in Korean Question Answering System*. (in Korean) In "Proceedings of the 12th Conference on Hangul and Korean Language Processing", pp. 184-189.

Lee K., Kim J. and Choi, K. (2000) *Construction of Test Collection for Evaluation of Question Answering System*. (in Korean) In "Proceedings of the 12th Conference on Hangul and Korean Language Processing", pp. 190-197.

Miller G. (1990) *WordNet: An on-line lexical database*. International Journal of Lexicography, 3/4.

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)", from http://trec.nist.gov/pubs/trec8/t8_proceedings.html

Prager J., Brown E. and Coden A. (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)", from 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, pp. 1022-1036.

TREC (Text REtrieval Conference) Overview. (n.d.) Retrieved April 22, 2002, from http://trec.nist.gov/overview.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. (2000) *Building a Question Answering Test Collection*. In "Proceedings of SIGIR 2000", pp. 200-207.

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