Enhancing Performance with a Learnable Strategy for Multiple Question Answering Modules

Hyo-Jung Oh, Sung Hyon Myaeng, and Myung-Gil Jang

A question answering (QA) system can be built using multiple QA modules that can individually serve as a QA system in and of themselves. This paper proposes a learnable, strategy-driven QA model that aims at enhancing both efficiency and effectiveness. A strategy is learned using a learning-based classification algorithm that determines the sequence of QA modules to be invoked and decides when to stop invoking additional modules. The learned strategy invokes the most suitable QA module for a given question and attempts to verify the answer by consulting other modules until the level of confidence reaches a threshold. In our experiments, our strategy learning approach obtained improvement over a simple routing approach by 10.5% in effectiveness and 27.2% in efficiency.

Keywords: Question answering, machine learning, strategy learning.

I. Introduction

Employing multiple question answering (QA) techniques for increased accuracy of an answer has been studied in past QA research. A naïve approach is to route questions to all the QA modules corresponding to the employed techniques as in a meta-search. This simply distributes an internally processed question to the individual QA modules in parallel and then merges the resulting answers. While most multi-technique based[1] systems [1]-[3] have adopted this straightforward strategy, an obvious weakness of this approach is an inefficient use of resources, especially with a large number of QA modules.

One can argue that the same answer from multiple sources will increase the confidence level [4]-[6]. Depending on the type of question and the nature of QA modules, however, this type of redundancy may not be necessary. For example, a question such as "When was Madam Curie born?" can be answered without ambiguity in an encyclopedia-based QA system, if an answer exists, because it can be handled by a pre-constructed knowledge base (KB). Besides, multiple answers from multiple QA modules may end up lowering the confidence level of the correct answer if a straightforward merging method is used. We argue that some redundancy is useful for answer verification but should be used more judiciously for both efficiency and effectiveness.

Another approach is to "hard-code" a merging strategy manually. When implementing a QA system, the designer has to understand the capability of individual QA modules and carefully craft the merging strategy for the given modules. This

Manuscript received July 10, 2008; revised Apr. 21, 2009; accepted June 9, 2009.

This work was supported in part by the Korea Ministry of Knowledge Economy (MKE) under Grant No. 2008-S-020-02 and by Brain Korea 21 project sponsored by Ministry of Education and Human Resources Development, Rep. of Korea.

Hyo-Jung Oh (phone: +82 42 860 5405, email: ohj@etri.re.kr) and Myung-Gil Jang (email: mgjang@etri.re.kr) are with Software & Content Research Laboratory, Daejeon, Rep. of Korea.

Sung Hyon Myaeng (email: myaeng@kaist.ac.kr) is with Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea.

doi:10.4218/etrij.09.0108.0388

[1] This is sometimes referred to as a "multi-strategy" in the literature, but the word "strategy" has a different meaning in this paper.

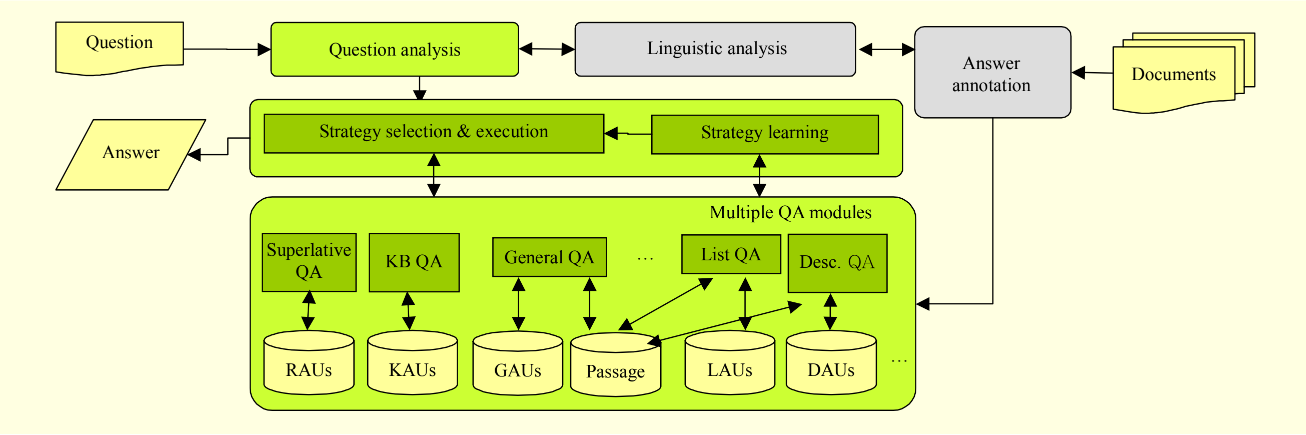


Fig. 1. System architecture.

method seeks to approximate domain-specific knowledge and identifies combining strategies developed by human users [7], [8]. QA systems based on this approach assign a suitable strategy for a user question based on some built-in rules or scenarios. However, such a QA system with a pre-defined strategy for combining the results makes it difficult to adjust to a new situation where a new QA technique or module is added. What is problematic is not only the initial cost for manual work but also the lack of extensibility and flexibility of the QA system.

This paper proposes a learnable strategy-driven QA approach that aims at enhancing both efficiency and effectiveness. A strategy is learned to determine the sequence of QA modules to be invoked and to decide when to stop invoking additional modules. The learned strategy invokes the most suitable QA module for a given question and attempts to verify the answers by consulting other modules.

II. Strategy-Driven QA

1. Overview

A strategy-driven QA system is assumed to have multiple QA modules as in the system shown in Fig. 1, which consists of six different modules. Given a question, the system determines a strategy to predict which QA modules are likely to find answers. The QA modules are then invoked sequentially to extract and verify the answers and to boost their confidence values if possible.

A user question in the form of natural language is entered into the system and analyzed by the *question analysis* component, which employs various linguistic analysis techniques, such as POS tagging, chunking, answer type (AT) tagging [9], and some semantic analysis, such as word sense disambiguation [10]. An internal question generated by the question analysis component has the following form:

$$Q = <AF, AT, QT, AS>,$$

where *AF* is the expected answer format, *AT* is the expected answer type, *QT* is the theme of the question, and *AS* is the information related to the expected answer source or QA module from which the answer is to be found.

- The answer format (AF) of a question is determined to be one of these four types: a *single, multiple, descriptive*, or *yes/no* question. For example, *single* is the AF value in the question "Who killed President Kennedy?."
- There are 147 fine-grained ATs organized in a hierarchical structure with 15 nodes at the level directly below the root, each of which has two to four lower levels [9]. The AT gives information about the type of the entity being sought [11]. The sub-type/super-type relations among the ATs give flexibility in matching. For the preceding example, the AT would be "people" because "who" can be matched with "president" in a passage.
- A question theme (QT) has two parts: a target and a focus. The target of a question is the object or event that the question is about, and the focus is the property being sought by the question. In the example above, the target is "J. F. Kennedy" and the focus is "killer."
- The answer source (AS) of a question indicates the most likely source (QA module) from which an answer can be found, which is determined based on the other traits of the question (AF, AT, and QT). It also contains some detailed information about what should be sought in the QA module. For example, "How can we prevent a cold?" and "How can we cure a cold?" are analyzed for an answer by the descriptive QA module but with additional information showing that the answer must be a method for something.

Among the four elements, AF can be determined relatively easily with the semantic class of the word after the interrogative in a question. AT, which is trickier to find, is determined by a hybrid classifier which combines maximum-entropy (ME)-based [9] and rule-based methods. The machine-learning-based method alone with relatively simple linguistic features was not sufficient to catch subtle nuances in Korean questions, necessitating the rule-based method that primarily relies on 1,113 lexico-semantic patterns (LSPs) [12] which we created manually. To determine QT, we devised a method that helps matching lexically different expressions by referring to a lexical database called the Korean Lexical Concept Net for Nouns (LCNN), which was manually constructed [11]. Lexically different predicates can be also matched by referring to the Korean Lexical Concept Net for Verbs (LCNV) [11]. AS is determined by the parse tree of a sentence, together with the three other elements of the question, AF, AT, and QT, which themselves are obtained from the same parse structure. To do this, we defined more than 1,500 patterns (for example, 161 patterns for superlative questions) organized into templates.

The *strategy selection and execution* component selects a strategy based on the internal query, invokes one or more QA modules depending on whether the calculated evidence for each answer candidate is strong enough, and finalizes the answer by incorporating answers and their evidence values returned from multiple QA modules if necessary.

The performance of the strategy-driven QA depends heavily on the accuracy of the question analysis because a strategy is selected based on the AS of the analysis result, which determines the QA module to be invoked first. Invoking more than one module can compensate any possible errors in the question analysis and in the answers from a QA module. In other words, answers from the first QA module are verified, and their confidence values are boosted if appropriate.

The *answer annotation* component builds heterogeneous answer bases for multiple QA modules: a learning-based method for a knowledge-based QA, a template-based method for a superlative QA, a sentence-pattern-based method for a list QA and descriptive QA, and a traditional statistical method for a general QA [13].

2. Multiple QA Modules

Multiple QA modules are tailored to various answer classes that are identifiable from documents. A new module that can be added as a new QA technique is developed for questions requiring a different answer class. While an AT refers to a named entity type being asked for in a question, "answer classes" are used to make a distinction among different traits of the answers, such as record, list, description, and general answer classes. In the current implementation, the QA modules represent six different answer classes.

In our document collection, unstructured text and structured/semi-structured data are mixed. Many sentences that appear there have particular structural patterns. Using information extraction (IE) techniques, these answers can be pre-acquired. We defined these answers as *knowledgebase answer units* (KAUs). For the second type, we focused on stereotyped sentences written in a *Guinness Book* style. Sentences including record information, such as "Mt. Everest was first climbed by Edmund Hillary," generally have specific words such as "first" which indicate that the sentence is a superlative sentence. We defined these answers as *record answer units* (RAUs). The example sentence, "Canada's official languages are English and French," is a chunk-type of list answer with parallel phrases. We defined this type of answer as *list answer units* (LAUs).

Another typical sentence type is the descriptive sentence, such as "A tsunami is a large wave, often caused by an earthquake" (X is Y). Because a corpus such as an encyclopedia or Wikipedia contains facts about many different subjects, or explains one particular subject in detail, there are many sentences that present definitions such as "X is Y." On the other hand, some sentences describe the process of a special event (e.g., World War I), so that they consist of particular syntactic structures (5W1H) similar to those found in news documents. We defined these descriptive sentences as *descriptive answer units* (DAUs). The other types are the *general answer units* (GAUs) and passage retrieval. These answers are retrieved in real-time based on the similarity calculation when a user question is entered, and are different from other answers.

While QA modules are complementary to each other in providing answers of different types, their answer spaces are not completely disjointed. For example, some factoid answers are found both in KAUs and GAUs. In other words, the GAU index is general enough to include terms in the KAU. This redundancy is a catalyst for answer verification by which answers from different modules boost their confidence levels among themselves.

III. Strategy Selection and Execution

Our QA framework using a learnable strategy makes use of a number of independent QA modules employing different answer finding methods. Each QA module in our system except a general QA is tailored to an answer class determined primarily by extractable and identifiable answers from documents, and by the nature of questions collected from the users. A strategy that determines the QA modules to be invoked when finding an answer is selected based on several factors such as the question's expected AF, AT, and AS corresponding to the expected answer class.

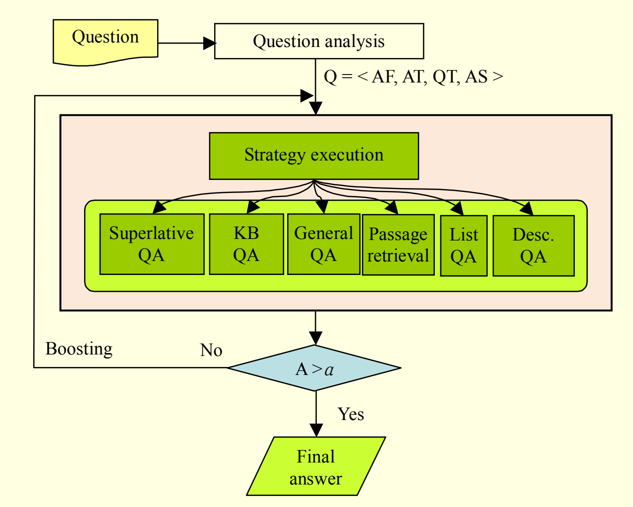


Fig. 2. Flow diagram of answer generation.

Table 1. Learned strategy.

Answer type	Answer format	Answer source	Strategy
146 single	Single	None	General QA + passage retrieval
		KB	KB QA → general QA + passage retrieval
		Superlative	Record QA → general QA + passage retrieval
	Multiple	None	List QA + general QA + passage retrieval
10 desc.	Descriptive	Definition	Descriptive QA (definition) → KB QA → passage retrieval
		Reason	Desc. QA (reason) → desc. QA (objective) → desc. QA function → passage retrieval
		Method	Descriptive QA (method) → Desc. QA (definition) → Passage retrieval ...

As shown in Fig. 2, the flow of strategy-driven QA proceeds as follows. First, a user question is analyzed in order to select an appropriate strategy consisting of a sequence of QA modules and associated threshold values. Then, the QA modules are invoked in sequence as in the strategy until the stopping condition is satisfied. Since the first QA module is where the expected answer class of the query is sought, the QA modules invoked later play a role of verifying and boosting the confidence levels of the answers identified by the first QA module. At the second iteration, the answers from the first QA module and those from the second QA module are merged to produce a new ranked list of answers, and the invocation of a third QA module, if any, plays the role of verification and boosting. This process goes on until the confidence level of the top-ranked answer exceeds the threshold associated with the first QA module, or until there is no more QA module to be invoked in the strategy, whichever comes first.

More precisely, once a strategy is selected, an internal question generated by the question analysis can be represented as the sequence

$$<q_1, q_2, \cdots, q_k>,$$

where k is the number of QA modules in the sequence of the particular strategy. Each q_i is in the form appropriate for the i-th QA module. When a ranked list of answers is returned from the i-th QA module, it is merged with the combined ranked list of answers up to the (i-1)th QA module, $A_{i-1} = <a_{i-1,1}, a_{i-1,2}, \cdots, a_{i-1,m}>$, to produce a new ranked list, $A_i = <a_{i,1}, a_{i,2}, \cdots, a_{i,n}>$, where a_{ij} is the confidence value for the j-th answer returned by the i-th QA module. Since the two lists are merged, $n \geqq m$ holds. The confidence value of an answer for a combined answer list is computed by first normalizing the values in A_i:

$$a_{ij} = a_{ij} / \max A_{i-1},$$

and by updating the confidence value for the answer that exists in both A_i and A_{i-1}:

$$a_{ij} = a_{i-1,j} + a_{i,j}/2.$$

Since our strategies are created with a stipulation that the answers from the earlier invocation in the sequence should be more respected (see section IV for details), the algorithm is called boosting. The boosting process basically reinforces the evidence of the answers already returned by a previously invoked QA module, although a new answer may be chosen as the top-ranked answer in the process. The answer verification and confidence boosting schemes are reflected in the normalization of the answers in A_i, and in the reduction of the confidence values of the common answers both in A_i and A_{i-1} by a half.

Table 1 shows an example of the learned strategies. The "→" symbol indicates the order of QA module invocations. Given two QA modules, QA_1 and QA_2, $QA_1 \rightarrow QA_2$ shows that QA_1 and QA_2 need to be invoked in sequence, whereas $QA_1 + QA_2$ indicates they are to be processed in parallel. If a question is determined to be answered by the KB QA module (the answer source being KB), the question is sent to the KB QA module first, and the five top-ranked answers are returned. If the top-ranked answer's confidence value computed by the module exceeds a predetermined threshold in the strategy, the answer becomes the final one. Otherwise, the results from the general QA module and the passage retrieval are merged and re-ranked for answer confidence boosting.

IV. Strategy Learning

Our main motivation for learning strategies for different types of questions, as opposed to manually constructing rules for selecting QA modules for a particular type of an AS of a question, is three-fold. First, selecting a single QA module for a

```
Input: training a set of $n$ <question, answer> pairs $m$ QA
        modules
Initialize (preparation):
    1. Build an $m \times n$ matrix whose element is a list of five
       answers returned from a QA module for a question.
    2. Divide the matrix into $mn_i \times m$ matrices ($M_i$'s), each
       corresponding to the subset of $n_i$ questions whose AS
       values are equal to a QA module.
Begin
  Do for $I$ =1, 2,$\cdots$,$m$ (i.e. for each AS using $M_I$)
      [Order QA modules and determine thresholds for
      questions of each AS type]
     Do for $J$=1, 2,$\cdots$,$m$
        1. Evaluate each QA model
        2. Order QA modules
        3. Compute threshold for each QA module
End
```

Fig. 3. Pseudo-code of the learning algorithm.

given question based on its AS value is not reliable because the question analysis component does not always select the best QA module, and the best QA module selected may not be self-sufficient to return the correct answer. Second, routing a question to all the available QA modules is neither efficient nor effective because some modules may end up providing an incorrect answer. Finally, while handcrafting a set of rules for selecting a sequence of QA modules is possible, it would be difficult to develop rules for predicting when to stop invoking additional QA modules and for combining pieces of evidence from different modules in a principled way.

If the learning task had been simply to build a classifier which maps a question to the most appropriate QA module or a ranked a list of modules, we could have employed one of the existing classification methods such as ME [10] or supported vector machine (SVM) [14]. However, it was not clear how a classification algorithm could be extended to include the task of threshold value setting for the sequence of selected QA modules, so this required us to devise a new algorithm.

For strategy learning, we used 260 pairs of training data, which were part of the entire set of 811 <question, answer> of various sorts in terms of answer sources and difficulty levels.

The following is an example of a <question, answer> pair:

```
<Original Q> Who is the inventor of the periodic law?
<Alt_Q > Name the creator of the periodic law
<Answer>
 <Ans> Dmitri Mendeleyev </Ans>
 <A_type> Person </A_type>
</Answer>
```

The pseudo-code for this strategy-learning algorithm is given in Fig. 3. In describing the algorithm, we assume 260 question/answer pairs in the training set and six QA modules without loss of generality.

1. Preparation

Each query in the training set is sent to all the QA modules to obtain the top five answers from each module. As a result, we can build a 6 × 260 matrix where an element is a list of five answers returned from one of the six QA modules.

This matrix can be divided into six small matrices, M_i, based on the AS value of each question. Let n_1, n_2,$\cdots$, n_6 be the number of questions in each matrix, where n_i is the number of questions determined to have the i-th QA module as the AS value:

$$M_i = \begin{bmatrix} a_{1,1}, & a_{1,2}, & \cdots, & a_{1,6} \\ a_{2,1}, & a_{2,2}, & \cdots, & a_{2,6} \\ & \vdots & & \\ a_{n_i,1}, & a_{n_i,2}, & \cdots, & a_{n_i,6} \end{bmatrix}.$$

Each column vector $C_{i,j}$ (i, j = 1, 2,$\cdots$,6) of M_i represents the top five answers for all the questions whose AS values are equal to the j-th QA module. Except for the column j equal to i, the answers in the other columns were obtained from QA modules that are different from the predicted AS.

2. Evaluation Step

We compare $C_{i,j}$ with the answers in the training set to see how good the answers returned from the i-th QA module are. For this, we first build a gold standard answer vector G_i from the training set

$$G_i = < g_1, g_2, \cdots, g_{n_i} >,$$

where g_{n_i} is the number of questions that were supposed to be sent to the i-th QA module. In essence, a list of answers (in this case, five) from $C_{i,j}$ is compared against an answer from G_i for all the elements in the vectors. In the pair-wise comparisons, we count the number of lists that contain the corresponding answer g_k (k = 1, 2,$\cdots$, n_i). This number is a measure of the goodness of QA module j, with respect to n_i questions whose AS values are the i-th QA. When j=i, the module is supposed to be invoked first for the n_i questions in the training set. We compute the goodness value for all six column vectors to order the QA modules in terms of the degree to which each module satisfies answer G_i. In a normal situation, that is, if the QA modules were built reasonably well, the QA module i must have the highest goodness value. The QA modules are ordered by the goodness values, and this sequence becomes an important part of the strategy for the particular AS.

3. Setting the Threshold Value

Given the set of questions Q_1 and the first QA module, QA_1, as well as its goodness value, which is a reflection of the ratio between the correct and incorrect answer lists, we compute a threshold value to be attached to the module in the strategy. We select the lowest and highest confidence values from the set of correct answers, CA_1, and from the set of incorrect answers, IA_1, respectively. Our rationale is that in order for an answer to be correct, its confidence value must be at least larger than the lowest confidence value among those of the correct answers in the training set. Similarly, in order for an answer to be judged incorrect, its confidence value must be smaller than the highest confidence value among those of the incorrect answers in the training set. A candidate threshold, t_1, is then computed as the middle point between the two values. Let the lowest confidence value from CA_1 and the highest value from IA_1 be 0.4 and 0.6, respectively. The threshold, t_1, can be 0.5.

In applying the strategy for a question, an answer whose confidence value is lower than the threshold should not be considered correct, and its confidence value should be boosted over the threshold with the help of other QA modules if possible in order to be accepted as the answer.

The initial threshold, t_1, is changed by considering the next QA module, QA_2, which can provide additional correct answers using its own threshold. By sending the questions with incorrect answer lists (Q_2) to QA_2 and receiving answer lists from it, we create the same situation with QA_1. The answers can be partitioned into correct and incorrect sets of lists, CA_2 and IA_2, the lowest and highest confidence values chosen, and the middle point obtained as threshold t_2. With this last example, let the lowest confidence value from CA_2 and the highest value from IA_2 be 0.5 and 0.6, respectively. The second threshold, t_2, can be 0.55. The only exception is that it is normalized as follows:

$$t_2 \leftarrow t_2 * (1 - t_1).$$

Then, t_2 is added to t_1 to produce a new t_1, making it tougher to exceed in an actual QA session. Having produced a new t_1, CA_1 is reexamined to form new sets of correct and incorrect answer lists, the lowest and highest values chosen, and the middle point obtained as a new threshold.

This process of boosting the threshold continues until there is no change in the threshold value or no more QA modules to be considered to finally set the threshold. By raising the threshold for a QA module using the normalized confidence values of the answers from the next QA module, a smaller number of answers can be returned from the first module in a QA session, which makes the returned answers more reliable. An answer whose confidence value is smaller than the threshold can be accepted if it is also returned by one of the next QA modules, and its confidence value is boosted. In essence, boosting a threshold value means that an answer must have a high enough confidence value from the first module or receive support from other modules.

Using t_2 as the first threshold for QA_2, the same iterative steps are applied to produce the final threshold for the module using the normalized confidence values of the answers from QA_3. The same process is applied for t_3 and so on. Note that the threshold values for new QA modules get smaller as they are added to the strategy, which increases the chance for later modules to return answers easily and hence support the previously extracted answers if they are also extracted again.

V. Evaluation and Analysis

1. Experimental Setup

From the outset of this research [13], our goal was to build a QA system that can handle a variety of types of questions and answers. Based on our analysis of 1,485 questions of various types collected from real users and their answers from the Web and an encyclopedia, we found that over 80% of the answers were obtainable from the encyclopedia, while the Web answers were sometimes contradictive among themselves and not always confirmative. Moreover, the encyclopedia answers were richer with fuller information in the articles concentrated on a topic. As such, we chose to use the Pascal^{tm} Encyclopedia (http://www.epascal.co.kr), currently consisting of 100,373 entries (articles) and 1,017,807 sentences belonging to 14 domains, such as “Person,” “Art,” and “Science.” The reliability and balanced diversity of information in the encyclopedia were deemed desirable for testing the proposed QA framework utilizing multiple QA modules.

For effectiveness comparisons, we employed a *mean reciprocal rank* (MRR) [15]. We also used *precision*, *recall*, and *F-score* with the well-known “top-5” measure, which considers whether a correct nugget is found in the top 5 answers. Like the TREC QA track [16], we have constructed various levels of question/answer types. A total of 311 question/answer pairs were used for building and tuning the system, and an additional 500 pairs were used for evaluation [13].

2. Experiment with Six QA Methods

With the goal of evaluating the proposed strategy-driven QA method, six cases were examined: (i) a traditional QA using general indexing and passage retrieval; (ii) one-best individual QA where a question was sent to the most appropriate QA module only; (iii) one-best QA with the traditional QA as a

Table 2. Overall comparisons.

	# of question	# of response	# of correct	Precision	Recall	F-score	MRR-5	Improvement (%) over				
								(i)	(ii)	(iii)	(iv)	(v)
(i) Traditional	500	401	286	0.713	0.572	0.635	0.507					
(ii) One-best	500	407	356	0.875	0.712	0.785	0.632	24.7				
(iii) One-best+ traditional	500	480	388	0.808	0.776	0.792	0.668	31.7	5.7			
(iv) Simple routing	500	500	395	0.790	0.790	0.790	0.684	35.0	8.2	2.5		
	Response time			2333.1 s (4.6662/question)								
(v) Manually combined	500	495	397	0.802	0.794	0.798	0.688	35.7	8.9	3.1	0.6	
	Response time			2142.3 s (4.2846/question)							8.91	
(vi) Strategy-driven	500	483	422	0.874	0.844	0.859	0.756	49.1	19.6	13.2	10.5	9.9
	Response time			1834.1 s (3.6682/question)							27.2	16.8

backup; (iv) simple routing QA where a question was routed to all the available QA modules and the answers were combined; (v) a manual combination of the available QA modules; and (vi) strategy-driven QA, the proposed method.

To see the value of the learned strategies, we optimized not only the four different QA methods (traditional, one-best, simple routing, and the strategy-driven) but also individual QA modules. The traditional QA used only statistical retrieval methods, namely, general QA and passage retrieval. The one-best QA system selects the best AS for the given question among multiple QA engines and then finds candidate answers in the particular QA module. Since the answer selection method applied to this case is exactly the same as that for the strategy-driven QA, the only difference is that one-best indivisual QA does not use any other QA modules than the best one. One-best with a traditional QA uses the traditional method as a backup to be used when the one-best method does not return any answer.

In the simple routing QA, all candidate answers from six QA modules were merged into a single ranked list with a linear combination of the weights after the confidence values were normalized so that all the answer lists would have the same confidence value ranges. To set the best situation for combination parameters, we used 571 QA pairs (260 for training as in section 4 and 311 for tuning) while assuming the six individual QA modules were optimized.

To see the value of automatic strategy learning, we created a case where the invocation sequence and the combination method were all handcrafted. The resulting rules define which QA modules should be combined selectively for each question type, rather than just sending the question to all the QA modules as in the simple routing case. For example, the superlative question should combine the superlative QA, the KB QA, and the general QA. The candidate answers were merged by the same method as in the simple routing QA.

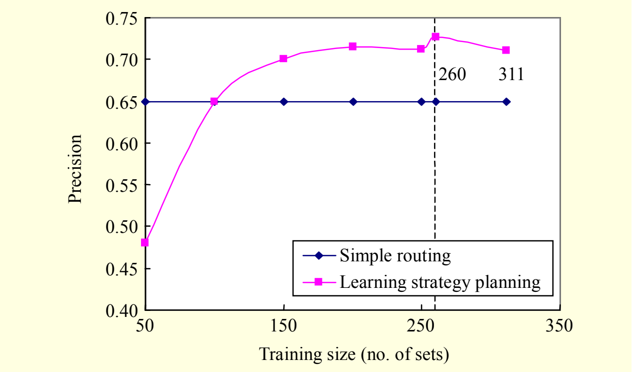


Fig. 4. Performance changes.

3. Experimental Results

As shown in Table 2, the final MRR values for the traditional (baseline) QA, one-best individual QA, one-best with a traditional QA backup, simple routing, manually crafted strategy, and strategy-driven QA are, 0.507, 0.632, 0.668, 0.684, 0.688, and 0.756, respectively. Our proposed strategy-driven method also shows the highest overall performance in both precision (0.874) and recall (0.844). There are 93 questions for which no answer was given from the one-best QA (only 407 responses as shown in Table 2). With the traditional QA backup, 73 questions receive candidate answers and 32 answers are correct among them. This demonstrates that combining multiple QAs (even only two modules) can achieve a better result.

In comparison with the simple routing method, which also makes use of the same multiple QA modules, the proposed method based on the strategy learning algorithm shows improvement of 10.5% (0.687 to 0.756 MRR) for effectiveness and 27.2% (4.667 s to 3.643 s per question) for efficiency. These results indicate that by executing only necessary QA modules with corresponding threshold values, we cannot only save time but also provide better answers, suppressing erroneous ones. Even compared with the manually crafted strategy-driven QA, our QA based on automatically learned strategies shows better results in both effectiveness (0.688 to 0.756 MRR) and efficiency (4.2846 s to 3.646 s per question). Needless to say, the cost of building a system manually is much higher than that of our learning method.

Meanwhile, the manually crafted rule-based QA shows a slight improvement in accuracy (0.6%, 0.684 to 0.688) over simple routing, while efficiency was improved by 8.91% (4.662 s to 4.284 s). However, the handcraft rules have to be revised when additional QA modules are introduced, which incurs much more additional cost.

In question answering, it is critical not to return incorrect answers to a user since returning an incorrect answer is usually worse than not returning any answer at all. Although it is challenging to explicitly recognize that there is no answer for a question, our proposed method ended up returning only 483 answers out of 500 (not returning 17) by enforcing the threshold values in the learned policy and not lowering the precision and recall values compared to the simple routing method. The traditional QA returned only 286 answers out of 500 questions, but the hard cut-off value across the board ended up reducing the recall value considerably to 0.572, while the precision was reasonably high.

For statistical significance of the results, we employed a *paired t-test* that assesses whether the means of two groups are statistically different from each other. The pair-wise t-test results for the MRR scores confirmed that the differences are statistically significant (p<0.0001).

Figure 4 shows how the effectiveness changed as we varied the size of the training data. As expected, as the training data increased, the performance improved quite dramatically from 50 to well beyond 150 <question, answer> pairs. We decided to stop adding training data when we saw a decrease in performance beyond 260.

4. Error Analysis

Since there are several components involved and several QA modules in answering a question, an error (incorrect answer) should be traced back to identify the very first place where the error occurred, that is, the cause of the error [17]. Table 3 defines possible sources of errors or factors in the system.

Based on the factors in Table 3, errors occurring in the experiment were analyzed. Table 4 shows the sources of 78 errors of the strategy-driven QA. Two of the four modules account for more than 80% (30.77% in the question analysis plus 48.72% in the answer selection) of the errors. The largest proportion of the errors was due to inaccuracy in the answer selection (48.72%), or more precisely, in the confidence value boosting (S2). Considering that only two errors occurred in the answer retrieval (R1), it is assumed that most candidate answers were retrieved. Instead, the cut-off value of 5 (meaning that only the top five documents were returned) was too small to include all the correct answers, aside from the errors incurred by the boosting process. In the question analysis module, the most critical factor was Q3, a failure in identifying appropriate expected answer types.

Table 3. Error analysis factors.

Component	Error type	Description
Question analysis	Q1	Linguistic analysis (POS tagging, parsing, WSD, etc.)
	Q2	Answer format analysis (single, multiple, descriptive, yes/no)
	Q3	Expected answer type analysis (location, date, etc.)
	Q4	Question target detection
	Q5	Semantic expansion of question theme (for example, predicate)
	Q6	Answer source analysis (KBQ, superlative Q, etc.)
Answer annotation	A1	Answer indexing (including knowledgebase construction process)
Answer retrieval	R1	Actual retrieval (answer retrieval, passage retrieval, etc.)
Answer selection	S1	Set 5 as the cut-off value
	S2	Confidence value boosting

Table 4. Error distribution in strategy-driven QA.

Component	# of error (%)	Error type	# of Q (%)
Question analysis	24 (30.77)	Q1	1 (1.28)
		Q2	4 (5.13)
		Q3	9 (11.54)
		Q4	5 (6.41)
		Q5	3 (3.85)
		Q6	2 (2.56)
Answer annotation	14 (17.95)	A1	14 (17.95)
Answer retrieval	2 (2.56)	R1	2 (2.56)
Answer selection	38 (48.72)	S1	9 (11.53)
		S2	29 (37.18)
Total			78 (100)

VI. Conclusion

The main motivation behind this research was to identify the best possible sequence of QA modules for an individual query. Instead of sending a question to all the modules and combining the results in an ad hoc way, we proposed a learning algorithm by which a set of strategies are learned for different types of questions. A strategy is a sequence of QA modules to be invoked and corresponding thresholds by which a decision is made on whether an answer is to be accepted. If no answer is accepted, the question is sent to the next module in the sequence. The confidence value from the first QA module is then boosted by the confidence value returned from the second module and so on. A novel method for strategy learning was also introduced.

To show the efficacy of the proposed model, we conducted experiments for four different cases. The proposed method obtained an improvement over the simple routing approach of 10.5% in effectiveness and 27.2% in efficiency. Even compared with manually built QA invocation rules, our strategy-learning-based QA shows better results in both effectiveness and efficiency.

A detailed error analysis shows that most errors were attributed to the question analysis and answer selection components. In particular, the expected answer type analysis for a question presents a critical problem because it influences the strategy selection process. We plan to improve the answer type classification method using hybrid machine learning algorithms. In addition, to overcome missing candidate answers in the answer boosting process, a dynamic cut-off scheme and a different boosting method will be developed.

References

[1] J. Chu-Carroll et al., "A Multi-strategy and Multi-source Approach to Question Answering," *Proc. 11th Text REtrieval Conference* (*TREC-11*), 2002, pp. 281-288.

[2] J. Lin and B. Katz, "Question Answering from the Web Using Knowledge Annotation and Knowledge Mining Techniques," *Proc. 12th Int. Conf. Information and Knowledge Management,* 2003, pp. 116-123.

[3] B. Katz et al., "Answering Multiple Questions on a Topic from Heterogeneous Resources," *Proc. 13th TREC*, 2004.

[4] A. Hickl et al., "Question Answering with LCC's CHAUCER at TREC 2006," *Proc. 15th TREC*, 2006.

[5] D. Moldovan, M. Bowden, and M. Tatu, "A Temporally-Enhanced PowerAnswer in TREC 2006," *Proc. 15th TREC*, 2006.

[6] E. Nyberg et al., "The JAVELIN Question Answering System at TREC 2003: A Multi-strategy Approach with Dynamic Planning," *Proc. 12th TREC*, 2003.

[7] A. Hickl et al., "Experiments with Interactive Question Answering in Complex Scenarios," *Proc. Workshop on the Pragmatics of Question Answering at HLT/NAACL*, 2004, pp. 60-69.

[8] H-J. Oh et al., "An Integration Approach of QA Engines Depending on Answer-Classes," *Proc. IEEE IRI*, 2006, pp. 178-181.

[9] C.K. Lee et al., "Fine-Grained Named Entity Recognition Using Conditional Random Fields for Question Answering," *Proc. AIRS*, vol. 4182, 2006, pp. 581-587.

[10] C.K. Lee et al., "A Multi-strategic Concept-Spotting Approach for Robust Spoken Korean Understanding," *ETRI Journal*, vol. 29, no. 2, Apr. 2007, pp. 179-188.

[11] M.R. Choi, J. Hur, and M-G. Jang, "Constructing Korean Lexical Concept Network for Encyclopedia Question Answering System," *Proc. IEEE IECON*, vol. 3, Nov. 2004, pp. 3115-3119.

[12] P.S. Jacobs, G.R. Krupka, and L.F. Rau, "Lexico-semantic Pattern Matching as a Companion to Parsing in Text Understanding," *Proc. Workshop on Speech and Natural Language in HLT Conference*, 1991, pp. 337-341.

[13] H.-J. Oh, *Compositional Question Answering with Collaborative Strategies*, PhD dissertation, Information and Communication Univ. Korea, 2008.

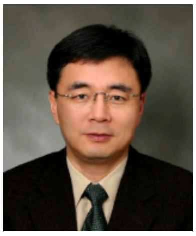
[14] C.K. Lee and M.G. Jang, "Fast Training of Structured SVM Using Fixed-Threshold Sequential Minimal Optimization," *ETRI Journal*, vol. 31, no. 2, Apr. 2009, pp. 121-128.

[15] E.M. Voorhees, "The TREC-8 Question Answering Track Report." *Proc. 8th TREC*, 1999, pp. 77-82.

[16] J. Lin. "Is Question Answering Better Than Information Retrieval? A Task-Based Evaluation Framework for Question Series," *Proc. HLT/NAACL*, 2007, pp. 212-219.

[17] D. Moldovan et al., "Performance Issuse and Error Analysis in an Open-Domain Question Answering System," *ACM Trans. Information Systems* (*TOIS*), vol. 21, no. 2, 2003, pp. 133-154.

Hyo-Jung Oh received the BS and the MS degrees in computer science from Chungnam National University, Daejeon, South Korea, in 1998 and 2000, respectively. She received the PhD degree in computer science from Information & Communication University (ICU) (now KAIST), Daejeon, South Korea, in 2008. Currently she is a senior researcher in Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea. Her research interests include machine learning, question answering, and knowledge mining.



Sung Hyon Myaeng is currently a professor at Korea Advanced Institute of Science and Technology (KAIST), Korea. Prior to this appointment, he was a faculty at Chungnam National University, Korea, and Syracuse University, USA, where he was granted tenure. He earned his MS and PhD from Southern Methodist University, Texas, USA, in 1985 and 1987, respectively. He has served on program committees of many international conferences in the areas of information retrieval, natural language processing, and digital libraries, including his role as a chair for ACM SIGIR, 2002 and 2008. He is on editorial boards of several international journals, including *ACM Transactions on Asian Information Processing* as an associate editor and *Information Processing and Management*. Recently he won an award from Microsoft Research, based on global competition for the RFP 'Beyond Search–Semantic Computing and Internet Economics.' His research interests include text mining, information retrieval, natural language semantics, and commonsense computing.

Myung-Gil Jang received the BS and MS degrees in computer science from Busan National University in 1988 and 1990, respectively. He received the PhD degree in computer science from Chungnam National University, Daejeon, South Korea, in 2002. Currently he is in charge of the Knowledge Mining Research Team, ETRI, Daejeon, South Korea. His research interests include natural language, information retrieval, question answering, and semantic web.