Optimizing Question-Answering Systems Using Genetic Algorithms

Ulysse Côté Allard[1], Richard Khoury[2], Luc Lamontagne[1], Jonathan Bergeron[1], François Laviolette[1], and Alexandre Bergeron-Guyard[3]

[1] Department of Computer Science and Software Engineering, Université Laval, Québec, Canada
{Ulysse.Cote-Allard.1, Luc.Lamontagne, Jonathan.Bergeron.6, Francois.Laviolette}@ift.ulaval.ca
[2] Department of Software Engineering, Lakehead University, Thunder Bay, Canada, Richard.Khoury@lakeheadu.ca
[3] Defence Research & Development Canada, Valcartier, Canada, Alexandre.BergeronGuyard@drdc-rddc.gc.ca

Abstract

In this paper, we consider the challenge of optimizing the behaviour of a question-answering system that can adapt its sequence of processing steps to meet the information needs of a user. One problem is that the sheer number of possible processing sequences the system could use makes it impossible to conduct a complete search for the optimal sequence. Instead, we have developed a genetic algorithm to explore the space of possible sequences. Our results show that this approach gives the system the adaptability we desire while still performing better than a human-optimized system.

1. Introduction

Question-answering (QA) systems are hugely popular in the scientific literature as well as in practical consumer applications. However, while there is undeniably a lot of diversity in QA algorithms, many of them suffer from a common weakness, which is that they try to design the single best sequence of processing steps to get from the question to the answer. Once implemented, such systems leave little to no room for customization of the algorithm to better address the information needs of a specific user.

In this paper, we start instead from a QA system that represents its processing steps as a set of filters applied sequentially, and which allows the modification of this sequence to meet the custom needs of users. This changes the QA design problem significantly: the challenge is no longer how to design new and better filters or processing algorithms, but becomes instead how to discover the best combination of the filters available to answer the user's questions. Indeed, given even a small number of filters and the fact that repeating and/or omitting filters in the sequence is acceptable, the number of possible sequences to consider quickly becomes massive and a complete search impossible. In this paper, we demonstrate how a genetic algorithm (GA) can be used to search the space of possible sequences to efficiently find combinations of filters that surpass the performance of human-designed sequences.

The rest of the paper is organized as follows. Section 2 presents the context of our research and an overview of the literature on QA systems with an emphasis on those that make use of GAs. We present in detail our implementation of the GA system in Section 3, and then we move on in Section 4 to describe and analyze our experimental results. Finally, we give some concluding remarks in Section 5.

2. Background

This paper is part of a larger research collaboration to develop an adaptive Intelligent Virtual Assistant (IVA), as described in (Lamontagne *et al.* 2014). The basic structure of this IVA is presented in Figure 1. In its intended mode of operation, the IVA would access the same incoming information reports as the user, annotate them and combine them with other information sources (such as a local collection of past reports and external resources of general background knowledge), and then fetch information customized to the user's needs as they work on preparing a new report. A central aspect of the system is thus the customizable question-answering (QA) system (Chinaei *et al.* 2014), which will query the information sources and return useful facts to the user. Moreover, this QA system must adapt itself to the needs of the user, as different users will have different information needs. For example, a user who has lived in a foreign country for several years will not ask the same questions or need the same answers about that nation as one who has never travelled there, and a user with an interest in national economic policy will expect different answers on questions about that country from one with an interest in grassroots social movements. This is the contribution of the user model component in the system.

A filter-sequence-based QA system such as Ephyra (Schlaefer *et al.* 2006) presents important advantages to a system such as ours, when compared to more traditional QA systems. A traditional QA system works in a predetermined sequence of steps, which typically cover first some query processing to narrow down the information requested by the question, then information retrieval to find a set of texts that may contain the answer, and finally an answer selection step to find the correct answer from the retrieved texts (Tomljanovic, Pavlic, and Katic 2014). The issue is that these steps are hardcoded into the QA system, and cannot be changed except by editing the software and recompiling it with a new (and still fixed) version of the sequence of steps. By contrast, a filter-sequence QA system has access to a library of algorithmic filters that can be used interchangeably and applied to data in any order. These filters can be general, such as a named entity retrieval algorithm, or they can be specific, such as a filter to boost the importance of geography-related answers for a user who needs help in geography. The advantage of a filter-based QA system in our system is its power for user-customization: by adding and removing filters and by changing the order of the filters in the sequence, the QA system will generate different answers from the same data for the same queries but that are appropriate for the different needs of different users.

One major problem with a filter-sequence QA system is selecting the optimal sequence of filters to use. Indeed, since filters can be arranged in any order in the sequence and will have different effects on the final result depending on what processing is done in the sequence before and after their contribution, the number of distinct sequences to consider is the factorial of the number of filters. For example, the 53 filters included with OpenEphyra (OE), a free open-source version of the Ephyra QA framework[1], can be arranged in 4.27×10^{69} distinct sequences. And when filters may be repeated multiple times in the sequence or omitted from the sequence altogether, the number of possible sequences becomes infinite. To make matters worse, there isn't a single best solution to be found, nor even a best solution per user, but an optimal solution given the specific information needs of each user as they prepare the current information report.

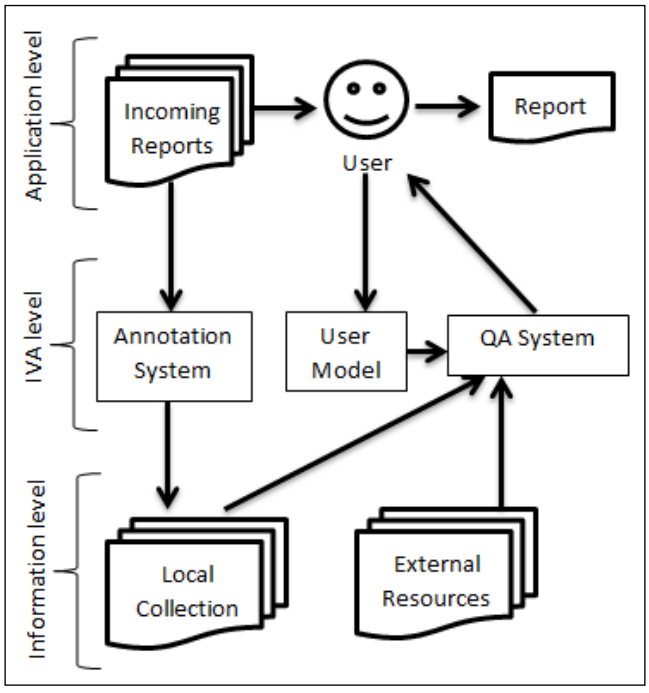

Figure 1: Structure of our IVA system.

Since thoroughly searching this massive and potentially infinite space for multiple different optima is not an option, we turn to a genetic algorithm to try to evolve good filter sequences which can retrieve good answers given a user's current information needs. A genetic algorithm represents solutions to a problem as individuals in a population, with features of the solution as genes making up each individual. New individuals can then be created through crossover (randomly combining the genes of two individuals) or mutation (randomly changing some genes in one individual). A fitness function is used to determine how good a solution is and to compare and rank solutions relative to each other. And finally, a roulette wheel is used to select individuals; this wheel is biased based on fitness, so that the best individuals at each generation have a greater chance of being selected for crossover, while the worst have a greater chance of being removed from the population.

GAs have been scarcely used in QA research. In (Samarakoon, Kumarawadu, and Pulasinghe 2011) for example, a GA was used to optimize the parameters of a ranking equation meant to score documents as answers to a specific query. This is an idea similar to ours – using GAs to optimize the internal workings of a QA system – but applied on a simpler problem: in this case the QA system and scoring function are already designed and fixed and the GA is used only to change the weights of components in the scoring equation, while in our system the GA is in charge of designing the sequence of operations in a core component of the QA system. GAs are also used in (Figueroa and Neumann 2008) to help discover and extract answers from snippets of text. However, that system uses an ordinary QA system to get relevant snippets of text that the answer may be found in, and only uses GAs to search the retrieved snippets for the exact answer. To our knowledge, GAs have not been used to the extent we describe in this paper in a QA system before.

Likewise, the challenge of designing a QA system that adapts to its user is one that has received little attention. The focus has been instead on determining user intent be-

[1] Available at: https://mu.lti.cs.cmu.edu/trac/Ephyra/wiki/OpenEphyra

hind individual questions based on the question's content alone, without any user information. Some examples can illustrate the wide range of research done in that direction. In (Zhang *et al.* 2009), question features such as keyword hypernyms and sentence structure are used to determine the answer type (definition, person, quantity, etc.) and answer attributes (expertise level, subjectivity level) intended by the user. In (Yoon, Jatowt, and Tanaka 2011), intent keywords are extracted from queries and used to find similar queries in a QA database, and the user's intent in inferred from the answers of these similar queries. And in (De and Kopparapu 2010), each keyword in the query is assigned a topic tag, and the query intent is determined using decision rules based on the set of tags it contains.

The fact that a QA system can benefit from user profile information has not gone unnoticed in the literature. However, this is often seen as an application of user profiles; if a profile is available, then it can be used in a QA system. This was done in a limited fashion in (Mishra, Mishra, and Agrawal 2010), where geographical location was provided by the user to improve the QA performance. In (Du *et al.* 2013), the connections between users in a social network guided answer recommendations. In both cases, the QA system was shown to benefit from the user information, but it remains information external to the QA system itself.

Additionally, some authors have been proposed to learn a user profile from user questions and their answers. One such system is presented in (Zhao *et al.* 2009): user questions are mapped to leafs in a topic ontology and used to measure user interest in specific topics. However, that information is never fed back to the QA system to customize it to the user or improve its performance.

3. Genetic Algorithm

As mentioned in the previous section, in this work we made use of the OpenEphyra (OE) QA framework. This software works in three steps. The first step consists of query generation. Given a user's question, it performs various pre-processing steps such as stemming, stopword removal, answer type classification, and query expansion, in order to generate a large set of semantically equivalent queries. In the second step, OE uses the queries to retrieve a set of potentially relevant text snippets from its document database (the entire information level in Figure 1). Finally, in the third step, the snippets are put through a sequence of filters in order to extract, combine, and rank potential answers and return the best one. It is this sequence of filters in this third step that our work focuses on optimizing using a GA.

Since OE includes 53 different filters, we designed the individuals in our GA to have 53 genes, where the value in each gene represents a filter and the entire genome represents an ordered sequential application of those filters. We also added a 54^{th} filter, an empty filter with no effect on the data but which allows the GA to create individuals that do not use all 53 real filters. Since filters can appear multiple times in a genome, this creates a search space of approximately 6.56×10^{91} possible individuals.

The population size is of 200 individuals. At each generation, at least 180 new children are created by crossover and mutation from the initial 200 individuals, and the 20 best individuals from the previous generation are added by elitist selection to them in order to create a new population. While the specific ratio of 90% new individuals in each generation was picked empirically, the decision to generate a lot more new individuals than are saved was made to explore a larger sample of individuals in a reasonable time.

The crossover operation is done by selecting two parent individuals using a roulette wheel selection, and creating two new children individuals where, for each of the 53 genes, one child has a 50% chance of getting the value from one parent, and the second child automatically gets the value from the opposite parent. Each of these two children then has a 10% chance of inversion. If this happens, two split points are randomly selected in the child, and the filter sequence is inverted between these points.

Each of the new children created in a generation has a probability of mutating; this probability is inversely proportional to the dispersion of the population calculated using the standard deviation of the population, so that a very diverse population suffers fewer mutations than a very homogenous one. Three mutation operators have been implemented in our system. The shift mutation shifts all genes down starting from a random point to the first empty filter, thereby deleting the empty filter at the end of the shift and freeing up the random starting position at the beginning; that free gene is assigned a random new filter value. The second is the move mutation operation, which selects a random set of genes and moves them to a new part of the genome, overwriting the genes at the destination and freeing up the original positions; these are replaced by random new filter values. Finally the swap mutation exchanges the values of two randomly-selected genes in the genome. Which of the mutation operations will be applied is decided randomly; the move and swap mutations have a 10% probability each of being applied, and the shift mutation has an 80% probability of being applied. The choices of mutation operations and their probabilities have been made experimentally, by using a GA to sort random sequences of 53 numbers to specific orders and using the Damerau-Levenshtein distance as a fitness function.

Finally, it is possible for the new population to be above 200 individuals by this point. To bring it back to 200, we use a roulette wheel selection biased to kill off the least fit individuals, with an elitism constraint that the single best individual cannot be selected for removal.

GA Fitness Function

The fitness function used to evaluate the individuals consists in asking 610 training questions and counting the number correctly answered. We make no difference between two individuals that correctly answer different sets of questions of equal number. Moreover, we made sure that the answer to each question does actually exist in the document database available to our implementation of OE.

A major benefit of this fitness function is its flexibility for customization, which is an important feature for our IVA system. Indeed, it is critical for us that the GA be able to find not one best filter sequence, but the best filter sequence for a given user with a given purpose. This fitness function makes this possible by simply varying the set of evaluation questions and answers. A user's interests would be reflected by having more questions in a certain topic, which will bias the GA optimization to a sequence of filters that works best for those questions. Likewise, a user with expertise in a given topic will have answers to questions that are more detailed or technical than a user who is a novice in that topic, and again this will bias the GA search and the resulting sequence of filters. This customized list of questions and answers will be included in the user profile component of the IVA, and can be built over time at minimal cost through regular usage of the IVA by including a user feedback component that will allow each user to mark queries that have been answered to their satisfaction. This is represented by the connection loop from the QA system to the user to the user model in Figure 1.

However, we immediately found that this fitness function was too slow to be of use: Evaluating the fitness of one population using 610 training questions took on average upwards of 20 minutes per individual on a regular PC, which makes the evaluation of entire populations and generations completely impractical. Consequently, we came up with two additional improvements to the function in order to speed up the evaluation of the individuals.

Referring back to our description of OE, we noted that the QA algorithm works in three steps, and only the third one makes use of the filter sequence the GA is modifying. The first two steps do not change, and will always return the same results for every individual. Our first improvement was thus to precompute the results of the first two steps for each training question, and to modify OE to use these precomputed results and skip the first two steps when evaluating the fitness of individuals.

The second improvement we made to the algorithm was a result of noting that many individuals had extremely low fitness, between 0% and 5%. Complete and precise evaluation of these individuals seems unnecessary, as they have almost no chance of being selected for crossover and are very likely to be removed from the population altogether in one generation. Given this, we implemented a subset evaluation of the individuals. The training questions were randomly divided into subsets of 61 questions, and an individual must get a certain number of correct answers in each subset to merit being evaluated on the next subset. This had the benefit of terminating the evaluation of bad individuals quickly to avoid wasting time on them, while focusing on a more complete evaluation of better individuals.

In the end, these two improvements reduced the average fitness evaluation time of an individual to below 3 minutes.

4. Experimental Results

The GA was implemented with an initial population of 200 individuals, and was run for 100 generations. Although the ultimate goal is to create customized training QA lists for each user's profile, at this stage of the work an initial training list of questions was created with the goal of optimizing the overall performance of the QA system. Customized QA lists that focus on specific topics or require more in-depth information will be developed in future work, and can then be easily substituted into our GA's fitness function. The current training set of questions is composed of:

- 100 questions on time (e.g. "when did the attack on Pearl Harbor take place?")
- 100 questions on locations (e.g. "where is Canada?")
- 100 questions on people (e.g. "what is the profession of Barack Obama?")
- 110 questions on descriptions (e.g. "what is a cat?")
- 100 questions on numbers (e.g. "how many moons does Jupiter have?")
- 100 questions of other types not in the first five (e.g. "what animal was domesticated by man to watch over flocks of beasts?")

As mentioned previously, the fitness function evaluation of an individual consists in trying to answer each training question using the individual's sequence of filters. The filters return a series of ranked answers, and the fitness function gives the individual a score based on the number of correct answers in the rankings. We studied two different approaches in that respect. In the first case, the fitness function considered the question correctly answered if the correct answer was one of the top-two ranked answers, and in the second case if it was one of the top-ten ranked answers. We will refer to these as the top-two experiment and the top-ten experiment in the discussion of our results.

Figure 2 illustrates the average fitness of the GA population and the specific fitness of the best individual at each generation in the top-two experiment. It can be seen that the algorithm converges quickly. After about 30 generations, the best individual has been found and the population has stabilized. There are still some fluctuations in fitness in the figure; these are a side-effect of the subset evaluation we described previously. As individuals become better,

they are tested on more subsets of questions, and individu-

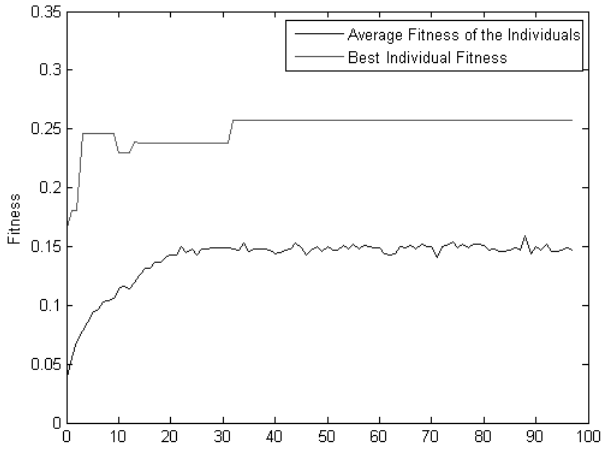

Figure 2: Fitness over generations of the GA population.

als whose high fitness were due to overfitting on one subset see performance drops and bring the population average down before they are taken out by selection. Note that only the top-two experiment was run for the entire 100 generations; the population in the top-ten experiment was found to have converged in 13 generations and terminated early.

At the end of evolution, the best individual of each population was selected as the best filter sequence our GA could find. Each individual was tested using a new set of 600 evaluation questions in the same six types as before. Each question was submitted to the QA system, and counted as correctly answered if the correct answer was in the top-2 or top-10 ranked results, and failed otherwise. These two tests match the two training conditions. For benchmark, we used the filter sequence recommended as best by the OE designers. The results obtained are presented in Table I and Table II for the test accepting answers in the top-2 results or in the top-10 results respectively.

The results demonstrate that both filter sequences exceed the performance of the benchmark on average. For specific question types, the best sequence from the top-ten experiment always surpasses the benchmark, and the best sequence of the top-two experiment equals or surpasses the benchmark in 9 of 12 statistics.

Table I: QA accuracy on top-2 answers

Question Type	OE Sequence	Top-two individual	Top-ten individual
Time	7%	4%	11%
Location	14%	11%	23%
People	4%	1%	9%
Description	4%	7%	12%
Number	9%	15%	15%
Other	3%	13%	25%
Average	**6.9%**	**8.5%**	**15.9%**

Table II: QA accuracy on top-10 answers

Question Type	OE Sequence	Top-two individual	Top-ten individual
Time	17%	17%	34%
Location	29%	30%	52%
People	8%	11%	26%
Description	7%	15%	26%
Number	17%	23%	31%
Other	16%	24%	53%
Average	**15.7%**	**20.1%**	**37.0%**

The tables also show that the best sequence from the top-ten experiment performs considerably better than its counterpart from the top-two experiment. This might be due in part to the random chance of evolution, but it might also be due to the more fine-grained fitness evaluation criterion of the top-ten experiment. Indeed, the top-ten evaluation makes it possible to compute and compare the fitness of less-performing individuals that return correct answers below the second position. For example, three individuals that return 300, 200 and 100 correct answers at ranks below 2 will receive different fitness values in the top-ten experiment, and the one returning 300 correct answers will be more likely to survive and be used in crossover while the one returning 100 correct answers is more likely to be killed off. On the other hand, in the top-two experiment, all three individuals will be evaluated equally with zero fitness, and all three will have equal probabilities of being used in a crossover or killed. Studying the impact of this finer evaluation is a direction for future research.

Next, we studied the sequences of filters evolved. There are some striking differences with the expert-designed OE sequence. The OE filter sequence is composed of 14 different filters, while the sequences discovered by our GA count 37 filters and 41 filters for the top-two and top-ten individuals respectively, and both show several duplicates. Moreover, the filters selected in the expert sequence and those in the evolved sequences highlight their different designs. The experts designed a sequence that mimics traditional QA systems, with clear sections to analyze the question, then to extract potential answers from documents, and then to score and rank the answers. On the other hand, the GA sequences are not designed with such preconceived notions, and do not present obvious logical sections and divisions like the expert sequence. It also includes filters with no equivalents in the expert sequence, such as the UnnecessaryCharactersFilter and the ResultLengthSorterFilter. This hints that the GA takes a more calculated approach to this task than the experts. Clearly, the GA studied sequences that are very different from the shorter and more traditional one OE experts recommended.

As noted, several filters are repeated in the GA sequences. For example, a filter to remove unnecessary characters from answer snippets is applied at three different

points in the top-two sequence. This might have occurred to remove unnecessary characters that appear after other filters pinpoint answer components. Or it could be a harmless result of the GA operators, since filters that have no effect on the answers do not negatively affect an individual's fitness. Future work may look at trimming these unnecessary filters from the sequence of the final retained individual, to minimize the execution time of the sequence.

We also note that our evolved filter sequences are both more than twice as long as the human-designed one; indeed, the ability to explore such longer sequences is one benefit of GAs. In our system, this is a result of the fact that the shift mutation, which accounts for 80% of all mutations, explicitly removes an empty filter, and that any new filter value added only has a 1/54 chance of being an empty filter. In other words, our system is biased towards creating longer filter sequences. Moreover, the results in Tables I and II seem to indicate that a filter sequence's performance is not directly correlated to its length; the top-two sequence is more than twice the length of the OE-recommended sequence and only four filters shorter than the top-ten sequence, yet its performance is not twice as good as the OE sequence and in fact is statistically closer to that of the much-shorter OE sequence than to the similar-length top-ten sequence. In future work, we plan on balancing the system by adding a predator (Li 2003) that targets genomes that represent longer sequences.

5. Conclusion

In this paper, we presented our work in designing and implementing a genetic algorithm to discover the optimal sequence of filters to apply in a question-answering system. The challenges faced include the massive search space that results from both the large number of filters available and the fact that the order in which they are applied affects the answers returned, and the unacceptably slow performance of the ideal fitness function we wanted to use in order to be able to introduce user customization later on. Our results demonstrate that the GA-optimized filter sequence outperforms the expert-designed sequence when it comes to answering a general set of questions.

Future work will introduce the notion of custom filter sequences in the GA, a central aspect of our IVA system. Since OpenEphyra already performs answer type classification as part of its preprocessing step, one level of customization that can be implemented immediately is to create filter sequences for each answer type. We expect this will lead to an important boost in the results, since different questions types have very different answers, and therefore could be extracted more efficiently using different filter sequences. Moreover, having different populations of filters will allow us to improve the GA by introducing migration (Cantu-Paz 1998), or the ability for selected high-fitness individuals to move from one population to another. This shares successful gene sequences discovered in one population with others, and help guide them towards new hybrid solutions that might not have evolved randomly.

References

Cantu-Paz, E. 1998. Survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systemes Repartis*, 10:141-171.

Chinaei, H., Lamontagne, L., Laviolette, F., Khoury, R. 2014. A topic model scoring approach for personalized QA systems. *Text, Speech and Dialogue: Lecture Notes in Artificial Intelligence* 8655:84-92. P. Sojka A. Horák I. Kopecek K. Pala eds.: Springer.

De, A., Kopparapu, S. K. 2010. A rule-based short query intent identification system, *Proc. of the 2010 International Conference on Signal and Image Processing*, 212-216, Chennai, India.

Du, Q. Wang, Q. Cheng, J., Cai, Y., Wang, T., Min, H. 2013. Explore social question and answer system based on relationships in social network. In *Proc. of the Fourth Int'l Conf. on Emerging Intelligent Data and Web Technologies*, 490-495, Xi'an, China.

Figueroa, A., Neumann, G. 2008. Genetic algorithms for data-driven web question answering. *Evolutionary Computation*, 16:89-125.

Lamontagne, L., Laviolette, F., Khoury, R., Bergeron-Guyard, A. 2014. A framework for building adaptive intelligent virtual assistants. In *Proc. of the 13th IASTED Int'l Conf. on Artificial Intelligence and Applications*, 17-19. Innsbruck, Austria.

Li, X. 2003. A real-coded predator-prey genetic algorithm for multiobjective optimization. In *Proc. 2nd int'l conf. on evolutionary multi-criterion optimization*, 207-221. Faro, Portugal.

Mishra, A., Mishra, N., Agrawal, A. 2010. Context-aware restricted geographical domain question answering system. In *Proceedings of the International Conference on Computational Intelligence and Communication Networks*, 548-553, Bhopal, India.

Samarakoon, L., Kumarawadu, S., Pulasinghe, K. 2011. Automated question answering for customer helpdesk applications. In *Proc. of the 6th IEEE International Conference on Industrial and Information Systems (ICIIS)*, 328-333. Kandy, Sri Lanka.

Schlaefer, N., Gieselmann, R., Schaaf, T., Waibel, A. 2006. A pattern learning approach to question answering within the Ephyra framework. *Text, Speech and Dialogue: LNAI,* 4188:687-694. P. Sojka, I. Kopecek, K. Pala, (eds.): Springer.

Tomljanovic, J. Pavlic, M., Katic, M.A. 2014. Intelligent question — answering systems: review of research. In *Proc. of the 37th Int'l Convention on Information and Communication Technology, Electronics and Microelectronics*, 1228-1233. Opatija, Croatia.

Yoon, S., Jatowt, A., Tanaka, K. 2011. Detecting intent of web queries using questions and answers in CQA corpus. In *Proc. of the 2011 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 352-355, Lyon, France.

Zhang, Y., Wang, X., Fan, S. Zhang, D. 2009. Using question classification to model user intentions of different levels. In *Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics*, 1153-1158, San Antonio, USA.

Zhao, Z., Feng, S., Liang, Y., Zeng, Q. 2009. Mining user's intent from interactive behaviors in QA systems. In *Proceedings of the First International Workshop on Education Technology and Computer Science*, 1025-1029, Wuhan, China.