Answering Complex Questions in Natural Language using Probabilistic Logic Programming and the Web

Jose Oramas M. [a] Luc De Raedt [b]

[a] *Katholieke Universiteit Leuven, ESAT - PSI*
[b] *Katholieke Universiteit Leuven, Dept. of Computer Science*

Abstract

We present an algorithm to answer complex questions in Natural Language that are a boolean combination of simple questions. The main feature of this algorithm is to use a probabilistic Prolog (ProbLog) to handle the uncertainty of answers obtained by information extraction systems such as TextRunner. The results shown in this paper indicate that the use of ProbLog improves the answers given by a system that parses complex questions with light linguistic mechanisms. Therefore, using a probabilistic setting can be seen as a promising approach for the enhancement of open information extraction systems based on weak linguistic algorithms.

1 Introduction

Relational Web search appeared as an improvement over traditional search, which was mainly driven by mechanisms based on keywords. Traditional search engines produced acceptable results but Web pages which did not contain the retrieved keyword(s) within its content or within its anchor text were not considered as candidate results. This search method seems to be semantically weak since it is only based on keyword occurrence rather than content "meaning". TextRunner [18] is a system that presents the potential of relational web search. On its first version, TextRunner scanned the content of 9,000,000 Web pages extracting approximately 11,000,000 facts. In addition, TextRunner is able to process simple queries in natural language. The candidate answers for a query in TextRunner are presented followed by the number of facts that support each answer. The higher the number of facts the more likely the answer. A weakness of this system is that it handles only queries that are formed by two entities and one relation. Complex queries cannot be processed and additionally, overly general queries may produce a wide variety of results. Given the abundance of information and the multitude of facts generated by TextRunner, some facts are more certain than others and thus it makes sense to associate a probability of certainty to each fact. There exist some probabilistic systems such as probabilistic Prolog (ProbLog) [4] and WHIRL [2] which allow the processing of massive data sets with attached probability values, and the discovery of relevant links between the entities on data sets. Based on the previous observations, this paper aims to answer the question as to whether the use of probabilistic mechanisms can enable the answering of more complex queries starting from probabilistic databases generated from systems such as TextRunner.

This paper is organized as follows: Section 2 introduces previous systems that address the answering of questions from automatic information extraction. The next section presents in detail our method for answering complex questions. Section 4 describes the test procedure that was used to evaluate the improvements given by the proposed algorithm. Section 5 presents the results of the evaluation followed by a discussion about the results, in section 6. Finally, conclusions and recommendations for future work are given.

2 Background

Some terms must be clarified in order to avoid confusion while reviewing the literature. In this article, the term *"query"* will be used to refer to a form of question in a line of inquiry represented by a formal notation.

The term *"question"* will refer to questioning forms represented in a natural language form. The task of information extraction has received some attention during the last years. A significant part of this work [3, 7, 12] can be grouped within the research line of *associative retrieval and spreading activation* [15], which is considered to be the retrieval of information with relevance to other information. These systems have as common characteristic their dependency on manual knowledge engineering and human feedback, which makes that they do not scale to the Web.

Within the line of Question-Answering (QA) systems the objective is to provide exact answers or sets of answers in response to queries. Following this QA paradigm, systems like BASEBALL [6] and LUNAR [17] which present accurate results, are limited to specific domains and depend on knowledge databases manually built by experts. More recent systems like [10], and *Ask.com* try to bring QA from fixed domains to the heterogeneous Web. These systems are based on heuristics used in linguistic transformations for question understanding and for answer extraction from a single matching sentence.

Information extraction is a line of research similar to QA. It aims to perform information extraction from Web pages in a unsupervised, domain-independent, large scale fashion. One of the first systems that followed this extraction paradigm is KnowItAll [5]. This system uses domain-independent sets of extraction patterns to learn domain specific extraction rules which are then applied to Web pages returned by external search-engines. DeWild [13] is a system in which queries are composed by text and wild cards. The results given by DeWild are in the form of unary, binary or general n-ary tuples. A query in DeWild search engine looks like a natural language sentence where the queried term or terms are replaced by wildcards giving the system the instruction to return possible terms found in the Web that might replace these wildcards. Similar to KnowItAll, DeWild also depends on external search engines to get results for the relaxed queries. In order to extract knowledge, TextRunner [1, 18] scans millions of web pages, analyzes the assertions within those web pages, and categorizes as facts the assertions with high number of occurrences. A query in TextRunner looks like a question expressing a relation between a given entity and the queried entity. As an output, TextRunner returns a list of possible answers with a number of facts supporting each answer. Different from KnowItAll and DeWild, TextRunner has the ability to handle large volumes of data without depending on external search engines, with better response speed as a consequence. The entity-relation representation, used in TextRunner, has a lot of potential considering the ongoing adoption of mechanisms as the resource description framework (RDF) and the web ontology language (OWL) for expressing the content of the Web. RDF is a data model in which statements about resources are done using a notation of the form subject-predicate-object. OWL is built on top of RDF with the objective of facilitating the interpretation of information by applications that want to process the information before displaying it.

The work previously mentioned is focussed on information extraction and answering of questions problems. A major focus has been set on developing algorithms that do not require supervision and that can scale to the Web. A second focus, that received less attention, refers to database interfaces that allow information extraction systems to be queried in a natural language mode. Apparently, little work has been done towards the answering of more complex queries. In this paper, we define *"complex queries"* as boolean expressions that can be chained by different operations stated in boolean logic. It must be noted that these boolean expressions are formal representations of simple questions which can be effectively answered by an existing system. From the systems mentioned above, DeWild is the only one that can answer complex queries to some extent. This is done by using more than one wildcard within the query sentence. However, DeWild's wildcards-based interface limits the extraction of information by constraining the text representation in which it can be available. Considering the features of these systems, this paper tackles the question as to whether the use of probabilistic mechanisms can enable the answering of more complex queries starting from probabilistic databases generated from one of these systems. Following this objective, and to be able to propose a domain-independent approach, we focus on information extraction systems. TextRunner is the selected system to be extended with probabilistic mechanisms. Different from DeWild, TextRunner does not require the retrieved information to follow a specific text representation, and allows more freedom on the way information is represented. Additionally, probabilistic facts can be derived from the results of TextRunner which are accompanied by a supporting number of facts. Recently, probabilistic mechanisms have shown good performance on the discovery of relevant links on massive biological databases [4, 16]. Therefore, the synergy of TextRunner with these probabilistic mechanisms proposes a promising foundation for extracting knowledge from a complex graph to the Web.

3 Proposed Approach

The question of the toy example *What is measured in pounds or carats and is used for jewelry?* is a complex question composed by the disjunction of the simple questions *What is measured in pounds?* and *What is measured in carats?* conjoined with the question *What is used for jewelry?*, which can be answered by TextRunner. However, answering these composed questions requires a more complex logical query than those that can be currently answered with TextRunner. For this reason there is the need to develop a module that can detect and parse these simple questions from complex questions, and that uses an appropriate method to combine the results obtained by each subquestion. Something important that should be noted is that the approach presented in this paper emphasizes the use of probabilistic mechanisms to solve the global problem of answering complex questions. Therefore, the use of complex and expert-assisted linguistic mechanisms is out of scope.

In order to enable the answering of more complex questions there are three subproblems that must be tackled. First, a mechanism for turning complex questions into simpler and easier to answer questions should be established. Second, TextRunner will be queried using the simple questions, and the obtained results will be processed in order to handle similar answers. Finally, once all the answers are available and accompanied by their respective probability, a probabilistic engine should be introduced to obtain the most probable answer for the question. The following algorithm briefly covers the steps taken to tackle the global problem.

STEP 1:	Run POS-Tagger
STEP 2:	Propose candidate subqueries based on pivot tokens
STEP 3:	Parse candidate subqueries using TALK program parser
STEP 4:	Query TextRunner with compliant questions
STEP 5:	Merge similar answers based on codebooks and TF-IDF
STEP 6:	Write the questions and answers in a formal relation language
STEP 7:	Compute a probability for each answer
STEP 8:	Generate a knowledge base
STEP 9:	Query the knowledge base using ProbLog

3.1 From Complex Questions to Answerable Simple Questions

To divide complex questions into simple questions the following algorithm is followed. First, the queried complex question is split into tokens, and Part-of-Speech (POS) tags are assigned to each token based on a maximum entropy model [8, 14]. POS tags mark words as corresponding to a particular part of speech based on their definition and context. In the example, the assignment of POS tags looks as follows.

Token		**POS tags**	**Token**		**POS tags**
What	:	WP, WDT, NNP	and	:	CC, NNP
is	:	VBZ	is	:	VBZ
measured	:	JJ VBD VBN	used	:	JJ, VBD, VBN
in	:	RB, RBR, FW, NNP, RP, IN	for	:	JJ, RB, RP, IN
pounds	:	NNS	jewelry	:	NN
or	:	CC	?	:	.
carats	:	NNS			

Based on POS tags, pivot tokens are defined. These pivot tokens consist of conjunctive and disjunctive terms which can serve as an axis to separate words or phrases, i.e. *“and”, “or”, “but”, “,”*, ... etc. Then, based on these pivots, candidate simple questions are defined.

Subquestion 1	pivot	**Subquestion 2**	pivot	**Subquestion 3**	
What is measured in pounds?	or	What is measured in carats?	and	What is used for jewelry	?

It might be the case that detected pivots are not necessarily representing conjunctions or disjunctions between their adjacent words or questions. For this reason, the candidate questions are parsed to check if they comply with the grammar of a simple question. This is realized using a parser for questions in English based on the TALK Program [11]. The TALK Program is written in Prolog where users can introduce facts in the form of simple assertions after which the program is able to answer questions related to those facts. This is achieved by routines for detecting if the chain of strings, entered by the user, has the structure of an assertion or question. The routines related to the assertions were removed while the routines for checking questions were extended with additional grammar elements to identify more variants of common simple questions. Once the candidate simple questions are parsed, compliant simple questions are sent as input to the next step in the process.

3.2 Extracting Results from TextRunner

Once the subquestions have been detected, they are used to sequentially query TextRunner. Each obtained result set is composed of a group of possible answers for every relation similar to the one queried in the question. Additionally, each answer is followed by the number of facts that support that answer. Referring to the example mentioned above, we assume the following results were given by TextRunner.

Subquestion 1 Answer (occurrences)	**Subquestion 2 Answer (occurrences)**	**Subquestion 3 Answer (occurrences)**
Pressure **(37)**	Diamond weight **(69)**	Gold **(53)**
Concrete strength**(26)**	Diamond **(31)**	Platinum **(27)**
Strength **(18)**	Gold **(30)**	Tiny diamonds **(20)**
Gold **(2)**	Gold Purity **(20)**	Diamond **(10)**
Impact strength **(2)**	Gemstone weight **(10)**	

In order to make these results more representative, similar answers must be merged. Considering the good results for this task presented in [2] a term frequency-inverse document frequency (TF-IDF) approach was applied. In this case, a vocabulary *T* was built using all the terms *t* on the answers given by TextRunner. The stopwords were removed and stemming was applied on the rest of the words. Based on this vocabulary, each of the answers were represented by a vector $\vec{v} = (v_1, v_2, ..., v_n)$, where each component v_t was computed based on the terms *t* within the answer. If a term *t* in *T* is not present in an answer represented by $\vec{v}$, then the component has a value of zero. Otherwise, the component value is computed as $v_t = (log(TF_{\vec{v},t}) + 1) \cdot log(IDF_t)$, where $TF_{\vec{v},t}$ is the frequency at which term *t* appears in the answer represented by $\vec{v}$, and IDF_t is the total number of answers divided by the number of answers that contain the term *t*. Once each answer has an associated vector, the similarity between answers represented by the vectors $\vec{v}$ and $\vec{w}$ is computed using the following formula $\text{SIM}(\vec{v}, \vec{w}) = \sum_{t \in T} (v_t \cdot w_t)/(\| \vec{v} \| \cdot \| \vec{w} \|)$. This formula can be interpreted as the cosine between two vectors. The range of similarity given by this formula is between 0 and 1. The closer the similarity value gets to 1, the more similar the answers represented by vectors $\vec{v}$ and $\vec{w}$ are. Once this value is computed between two vectors from an existing and a candidate answer, candidate answers with a similarity above a certain threshold are considered similar. This threshold was heuristically set to 0.5 to provide low filtering but the tuning of this parameter may lead to better results. If the similarity value is above the threshold, the candidate answer is considered a new text representation of the existing answer and it is not introduced in the graph. The number of facts associated to the new representation of the answer are added to the similar answer. By doing this, the number of total occurrences of a certain answer within the set is maintained.

This process is done using every result set obtained as an output from querying TextRunner in the previous step, presented in section 3.1. By applying the merging of answers in the example introduced above, answers like 'Impact Strength' and 'Strength' which have a high similarity value are merged. At the end we might expect the following results.

Subquestion 1 Answer(occurrences)	**Subquestion 2 Answer(occurrences)**	**Subquestion 3 Answer(occurrences)**
Pressure **(37)**	Diamond **(100)**	Gold **(53)**
Strength **(46)**	Gold **(50)**	Platinum **(27)**
Gold **(2)**	Gemstone weight **(10)**	Diamond **(30)**

3.3 A Probabilistic Framework

Given the abundance and multitude of answers produced by TextRunner, some answers may have a higher certainty than others. Therefore it is suggested to put these results, now weighted by number of occurrence, in a probabilistic setting. Another way to interpret the output of TextRunner, is to see it as a set of facts/assertions that answer a question. These facts can be represented as $r(e_1, e_2)$ where entities e_1 and e_2, which are usually noun phrases, are related by a relation *r*, which is usually an action. Based on this representation of results, a weighted graph is built where nodes are the entities participating on a relation and the links of the graph represent the relation between the linked entities weighted by the number of supporting facts.

Once the graph has been built, probability values must be assigned to each relation within the graph. These probability values are computed based on frequency of occurrence of the entities within a relation with a criteria that can be summarized with the formula $(r(v, w)) = \sum_{x_i} (n(r(x_i = v, w))/n(r(x_i, w))), r \in R$. For a relation *r* from the set *R* of relations within the graph, the probability assigned to *r(v,w)* is expressed

as the number of total facts *n(r(v,w))* supporting the relation *r(v,w)* divided by the number of total facts supporting the relation *r*. Note that in this formula we only consider the variations on *v* since it represents the queried entity of the relation. In the question answering task it can have the value of the candidate answers for a question.

Once every relation between nodes in the graph has been assigned a probability value, a knowledge base of probabilistic facts is built. TextRunner output, natural language assertions supported by a number of facts (NLAF), will be converted to probabilistic relational facts (PRF). In the example, this step can be represented in the following way.

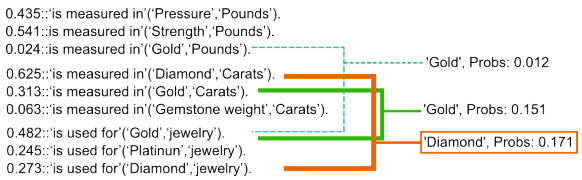
NLAF	**PRF**
Pressure **(37)**	0.435::'is measured in'('Pressure','Pounds').
Strength **(46)**	0.541::is measured in'('Strength','Pounds').
Gold **(2)**	0.024::is measured in'('Gold','Pounds').
Diamond **(100)**	0.625::'is measured in'('Diamond','Carats').
Gold **(50)**	0.313::'is measured in'('Gold','Carats').
Gemstone weight **(10)**	0.063::'is measured in'('Gemstone weight','Carats').
Gold **(53)**	0.482::'is used for'('Gold','jewelry').
Platinum **(27)**	0.245::'is used for'('Platinum','jewelry').
Diamond **(30)**	0.273::'is used for'('Diamond','jewelry').

3.4 Querying the Knowledge Base with ProbLog

Probabilistic Prolog (ProbLog) [4] is used to handle the probabilistic facts. ProbLog is a probabilistic programming language based on simple logic integrated with YAP (Yet Another Prolog) Prolog. It is essentially an extension of Prolog where facts are labeled with a probability that they belong to a randomly sampled program, and these probabilities are mutually independent. A ProbLog program thus specifies a probability distribution over all its possible non-probabilistic subprograms. The success probability of a query is defined as the probability that it succeeds in such a random program. Each time a user queries the system, a knowledge base is dynamically built with the results of the respective simple subqueries. The complex query is also written in a relational probabilistic format as can be seen in the following example.

Relational Representation
:- ('is measured in'(X,'pounds') ; 'is measured in'(X,'carats')) , 'is used for'(X,'jewelry').

When the query succeeds, ProbLog outputs the success probability accompanied by the value of X which satisfies the query. In short, the most probable answer will be selected in the presence of a set of possible answers. In our toy example, the selected answer would be 'Diamond' since is the answer with higher probability.



The decision of using ProbLog as the probabilistic mechanism to drive the extraction of probable answers was made based on the good results that it obtained in the mining of complex biological networks. In biological networks as large as the entire BIOMINE Network [16], with around 1,000,000 nodes and 6,000,000 edges, ProbLog processed queries in approximately 70 seconds [9]. This evidence makes ProbLog a promising candidate for the task of scaling up to large graphs extracted from the Web.

4 Testing

An experiment was performed to evaluate the effectiveness of the introduced probabilistic mechanisms for answering complex questions. This experiment aimed to answer the following experimental questions: *Question 1 (Q1): What are the improvements given by the probabilistic mechanism on the accuracy of the system?* and *Question 2 (Q2): How is the recall of the system affected by the introduction of probabilistic mechanisms?* The experiment consisted of answering 26 complex (52 different simple) questions. The

26 questions were gathered from 13 individuals from different fields, where each individual provided two questions. An example of the questions provided by the individuals is: *who proposed the theory of relativity and won a nobel prize?*) A requirement set on the questions was that the answer for the question had to be known. Once the questions were gathered, they were given as input to two variants of a QA system based on TextRunner. These systems receive as input candidate simple questions based on pivot tokens. These candidates were parsed using a variation of the TALK Program and were given as input to TextRunner. A knowledge base is built using the output from TextRunner. Finally, this knowledge base is queried in order to obtain an answer. The main difference between these two variants is the format in which they build the knowledge base and the means they use to query it. The first variant, the prototype system, builds a ProbLog knowledge base where probabilities are assigned to the answers based on occurrences reported by TextRunner and this knowledge base is queried with ProbLog. The second variant, the baseline system, builds a pure Prolog knowledge base, thus, no probabilities are assigned to the answers. Prolog is used to query the knowledge base of the baseline system

5 Results

From the total set of 26 questions only 23 were answered by both systems while for the other three questions no answer was found. The answers given by the two systems were compared to the expected answer. In this set, the baseline system was accurate 70% of the time, while the prototype system was accurate 91%. Answering *Q1* in this brief experiment, the use of a probabilistic mechanism provided an improvement of 21% on the accuracy of the system. Additionally, in order to evaluate the accuracy of the reported answers, the probability of the answers given by the baseline system were computed and compared with the probability of the answers given by the prototype system (fig.1.a). Once the baseline system built its knowledge base, the probabilities of the answers were computed following the same procedure as in the prototype system (section 3.3). It should be noted that for the baseline system these probabilities are only considered for the comparison of the results.

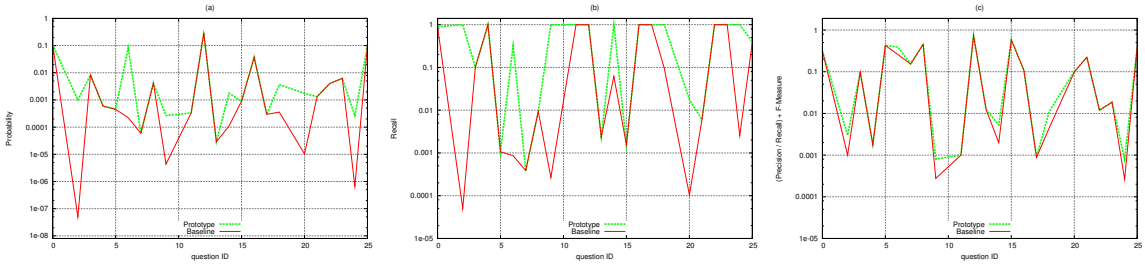


Figure 1: Comparison of (a) probabilities, (b) recall and (c) Precision/Recall + F-Measure of the answers obtained by the prototype system (green) and the baseline system (red).

As can be seen in fig.1.a, the performance of the prototype gives better results (with higher probability) than the results given by the baseline system. Fig.1.a also shows that both systems reported the same behavior (reported similar answers) in 83% of the questions. Addressing *Q2* to estimate the recall during the test, the knowledge bases of both systems were checked, as they contain all the possible answers and number of facts supporting each answer, that are considered for answering a certain question. There may be in these knowledge bases answers that are similar to the one reported by the system. Based on these answers, the recall is estimated as the number of facts considered by the system to report an answer divided by the number of facts available for that answer in the knowledge base. The later include number facts linked to the text representation of the reported answer plus the number of facts linked to similar answers. To measure the effectiveness of the retrieval, the F-Measure (F_1) was computed given the same importance to the precision and recall ($F_1 = 2((Precision \cdot Recall)/(Precision + Recall))$). The values of $F_1 + Precision/Recall$ on the performance of both systems were compared. As can be seen in fig.1.c the retrieval effectiveness of the prototype system is never below the effectiveness of the baseline system.

6 Discussion

After performing the experiments previously mentioned, the following observations are made.

While analyzing the knowledge bases it was found that the report of similar answers by 83% of questions was caused by two reasons. Either the reported answer was the only answer that satisfied the query or the answer with highest probability was located first on the list of answers satisfying the query.

The low values of recall, presented on some of the questions, were not caused by the criteria of measuring similarity between answers, but by the criteria used to select the most meaningful text representation for a group of similar answers. It was also detected that this is the reason why the systems could not find an answer for three of the questions. Future improvements must address this problem using a representation with more occurrences, or consider a bank of possible representations, and measure the similarity considering all of them. Referring to *Q2*, based on the way probabilities are assigned, we can say that the introduction of probabilistic mechanisms should increase the recall of the system (fig.1.b). However, this depends on the effectiveness of the merging and representation of similar answers.

"What is the gain of using the proposed (probabilistic) algorithm compared to another (non-probabilistic) algorithm specialized on computing the intersection of answers?". There are two main advantages of using the proposed algorithm: a) the possibility of handling uncertainty between answers and b) the different kind of questions that can be processed. In the presence of many answers the probabilistic algorithm selects the answer with higher accuracy and recall. This was found by obtaining an improvement of 21% on the accuracy over a system which did not use probabilistic means (fig.1). The specific intersection algorithms will have better performance only in sets with a fixed number of subqueries and relational operators where answers for these subqueries are unique. The use of ProbLog represents an advantage for answering large complex queries that produce a large number of possible answers. For example, if instead of generating the knowledge base dynamically, we had access to the TextRunner's knowledge base directly. Finally, assuming that the specific intersection algorithms are extended to handle likelihood of the candidate answers; it has to be noted that answers are boolean expressions and that conjunction is only one of the different logic operations that can be applied to these expressions. Given this observation, ProbLog gives us the advantage of handling all possible boolean logic operations that can be applied to these expressions.

The presented algorithm has some similarities and differences with the WHIRL system. Both methods combine text representations with logic representations. Both use weighting schemes to rank facts where the proposed algorithm uses probabilities, WHIRL uses numeric scores. As differences, in the proposed algorithm the similarities between answers are computed while the answers are extracted from TextRunner, while in WHIRL the similarity is computed on the fly at query time. A major difference between the two methods is that for WHIRL based systems all the relevant facts should be extracted before any query can be issued. In the case of the proposed algorithm, these facts are extracted dynamically just after the query is issued. Another major difference with WHIRL is that it is queried using a formal query language whereas our system is queried in a natural language form.

The examples presented in this paper only covered the answering of complex questions composed by the conjunction of two subquestions. It should be clear that it can also cover the conjunction of a large number of queries. To enhance the prototype for handling listing questions, instead of reporting only the most probable answer, the answers satisfying the question can be ordered by probability value and reported in that order. To cover other boolean operations, further modifications have to be done in the natural language interface of the system. The TALK program must be extended to cover the structure of a wider variety of questions and more terms of interest must be considered depending on the type of question.

7 Conclusions

"Is it possible to answer complex queries by means of probabilistic mechanisms?" This paper shows that the use of probabilistic means improves accuracy and recall on the answering of complex queries compared to non-probabilistic methods. In a context where light linguistic mechanisms are employed, less accurate results with different levels of certainty can be produced. In such a context probabilistic approaches play a more significant role, handling the uncertainty and mining the results for the most probable answer. As a consequence, probabilistic logic programming might represent a promising option to take open information extraction systems to a next level. In order to improve the results presented in this paper, three points must be addressed. First, a criterion for the selection of the most meaningful textual representation must be adopted, which will improve the recall of the algorithm. Secondly, different kinds of complex questions should be introduced. Finally, the testing of future prototypes following this algorithm should be done on a larger number of complex questions.

References

[1] M.J. Cafarella, M. Banko, and O. Etzioni. Relational web search. *UW CSE Tech Report*, 2006.

[2] W.W. Cohen. Whirl: A word-based information representation language. *Artificial Intelligence*, 118(1-2):163 – 196, 2000.

[3] Michal Cutler, Yungming Shih, and Weiyi Meng. Using the structure of html documents to improve retrieval. In *USITS'97: Procs. USENIX Symposium on Internet Technologies and Systems*, pages 22–22, Berkeley, CA, USA, 1997. USENIX Association.

[4] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: a probabilistic prolog and its application in link discovery. In *Procs. 20th Int. Joint Conf. on Artificial Intelligence*, pages 2468–2473. AAAI Press, 2007.

[5] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in knowitall: (preliminary results). In *WWW '04: Procs. 13th int. conf. on World Wide Web*, pages 100–110, New York, NY, USA, 2004. ACM.

[6] Bert F. Green, Jr., Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conf.*, pages 219–224, New York, NY, USA, 1961. ACM.

[7] Kaoru Hiramatsu, Jun-Ichi Akahani, and Tetsuji Satoh. Two-phase query modification using semantic relations based on ontologies. In *Procs. of IJCAI-03 Workshop on Information Integration on the Web*, pages 155–158, 2003.

[8] Heyan Huang and Xiaofei Zhang. Part-of-speech tagger based on maximum entropy model. *Computer Science and Information Technology, Int. Conf. on*, 0:26–29, 2009.

[9] Angelika Kimmig, Vítor Santos Costa, Ricardo Rocha, Bart Demoen, and Luc De Raedt. On the efficient execution of problog programs. In *Procs. 24th Int. Conf. on Logic Programming*, pages 175–189, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Cody Kwok, Oren Etzioni, and Daniel S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001.

[11] Fernando C. N. Pereira and Stuart M. Shieber. *Prolog and natural-language analysis*. Center for the Study of Language and Information, Stanford, CA, USA, 1987.

[12] Amit Pisharody and Howard E Michel. A search engine technique using relation based keywords. In *Procs. 2005 Int. Conf. on Artificial Intelligence, IC-AI 05*, 2005.

[13] Davood Rafiei and Haobin Li. Data extraction from the web using wild card queries. In *CIKM '09: Procs. 18th ACM conf. on Information and knowledge management*, pages 1939–1942, New York, NY, USA, 2009. ACM.

[14] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging, 1996.

[15] G. Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill, New York, 1968.

[16] Petteri Sevon, Lauri Eronen, Petteri Hintsanen, Kimmo Kulovesi, and Hannu Toivonen. Link discovery in graphs derived from biological databases. In *Procs. Int. Workshop of Data Integration in the Life Sciences*, pages 35–49, 2006.

[17] W. A. Woods. Progress in natural language understanding: an application to lunar geology. In *AFIPS '73: Proc.National computer conf. and exposition*, pages 441–450, New York, NY, USA, 1973. ACM.

[18] A. Yates, M.J. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. Textrunner: open information extraction on the web. In *NAACL '07: Procs. of Human Language Technologies*, pages 25–26, Morristown, NJ, USA, 2007. Association for Computational Linguistics.