Vietnam Journal of Computer Science
Vol. 7, No. 3 (2020)

DOI: 10.1142/S2196888820500190

NLP Questions Answering Using DBpedia and YAGO

Tomasz Boiński*, Julian Szymański†, Bartłomiej Dudek‡, Paweł Zalewski§, Szymon Dompke¶ and Maria Czarnecka‖

Department of Computer Architecture
Faculty of Electronics, Telecommunication and Informatics
Gdańsk University of Technology, 11/12 Narutowicza Street
Gdańsk 80-180, Poland
**tomasz.boinski@pg.edu.pl*
†*julian.szymanski@pg.edu.pl*
‡*bartlomiej.dudek@student.pg.edu.pl*
§*pawel.zalewski@student.pg.edu.pl*
¶*szymon.dompke@student.pg.edu.pl*
‖*maria.czarnecka@student.pg.edu.pl*

Received 28 December 2018
Accepted 9 April 2020
Published 8 June 2020

In this paper, we present results of employing DBpedia and YAGO as lexical databases for answering questions formulated in the natural language. The proposed solution has been evaluated for answering class 1 and class 2 questions (out of 5 classes defined by Moldovan for TREC conference). Our method uses dependency trees generated from the user query. The trees are browsed for paths leading from the root of the tree to the question subject. We call those paths fibers and they represent the user intention. The question analysis consists of three stages: query analysis, query breakdown and information retrieval. The aim of those stages is the detection of the entities of interest and its attributes, indicating the users' domain of interest. The user query is then converted into a SPARQL query and sent to the DBpedia and YAGO databases. The baseline and the extended methods are presented and the quality of the solution is evaluated and discussed.

Keywords: Question answering; NLP; YAGO; DBpedia.

1. Introduction

Research on automatic question answering systems started in the late sixties.[1] Over the years, the systems became more and more complex while often producing very good results.[2,3] Despite spectacular successes they are still far from human-like

*Corresponding author.

competences. Currently, most of the approaches are based on keywords, where the answer is derived directly from the specified by the user keywords and can be either an explicit answer or (usually) the set of documents containing the keywords from the query (and potentially containing the answer). In the latter case, the results obtained in this way are quite good; however, they require additional user verification and lookup within the documents provided. This approach is also counter-intuitive and only works when the database actually contains the question alongside the answer.

The situation differs when questions are formulated in natural language. In this case, there are no explicitly given keywords. The queries usually are ambiguous due to indirect subjects or lack of context. In this case, many systems rely on identifying interrogative pronouns to detect the entity of interest. Such an approach however does not work well for common-sense knowledge questions like "How many legs does a dog have?".

The questions formulated in natural language vary in difficulty depending on their complexity and ambiguity. Dan Moldovan *et al.*[4] defined five classes of question difficulty.

(1) The first class consists of factoid questions, where the answer usually can be directly found in the database ("When did Beethoven die?").
(2) The second class requires some knowledge about the question and the database structure. In this case, the answer might not be syntactically close to the question ("Who is the spouse of Grover Cleveland?").
(3) Class 3 of questions requires reasoning that is based on multiple, not always compatible sources.
(4) Class 4 of the questions are interaction-based.
(5) Class 5 questions that require expert systems able to analogical reasoning.

In this paper, we present an approach for answering questions formulated in natural language focused on the first two classes. The proposed method aims at answering questions from classes 1 and 2, providing, wherever possible, answers found in two databases: DBpedia and YAGO databases. Most of the other solutions use their own, dedicated knowledge bases and either are complicated systems or provide a list of potential answers for the user to choose from. The YAGO and DBpedia databases are vast and constantly improving, so we decided to base our solution on these two sources. The paper also aims at evaluating the solution against the well-known TREC question database.

The structure of this paper is as follows. Section 2 describes different approaches to question answering and our previous evaluation solution that served as a baseline for our approach. Section 3 presents the knowledge bases used in our approach. Sections 4 and 5 present in detail our approach, and shows improvements introduced to the algorithm. In Sec. 6, evaluation of the method is given. Finally in Sec. 7, we discuss the results and conclude.

2. Existing Solutions

Over the years, many interesting approaches to question answering emerged.[5,6] The first systems, like PRECISE[7] or BASEBALL,[8] focused on providing natural language interfaces to databases, mapped user queries to SQL queries.

Further, question answering systems were proposed on a selected open domain. Here, the approaches were not fine-tuned towards a specified domain, and needed to give answers to general questions. Many of the solutions were prepared during the TREC conference.[9] The best of those systems could answer as many as 70% of the questions from the TREC database,[10] ranging from simple factoid questions to complex, indirect questions. Some like LASSO,[4] used deep lexical analysis of the question, to provide the answer using an iterative process. Others, like QRISTAL[11] or QALC,[12] were based on semantic similarities and usually map the question into triples/queries. These approaches differ also in terms of complexity, from knowledge-rich systems[13] to simple systems like AskMSR.[14]

Another group of systems are ontology-based solutions. Such systems take queries given in natural language and, based on the used ontology they return the answer from one or more knowledge bases that are compatible with that ontology. Examples of such systems represent a very broad spectrum of solutions. Some, like SQUALL,[15] SPARQL2NL[16] or ORAKEL[17] convert the question into a SPARQL query that is evaluated against the given knowledge base.

In recent years, one very successful project was built, bringing baseline for the state of the art systems aiming at answering questions. IBM Watson[18] was developed as part of the DeepQA Project, which started in 2007. Its first implementation was made using a cluster consisting of around 2500 CPUs, 15 TB of RAM and, without a connection to the Internet. The quality of the system was so good that it managed to win the Jeopardy TV show.[19] Its strength comes from multiple algorithms that cooperate to calculate the best answer. The drawback of this solution is its complexity and limited availability for the wider audience.

In our approach, we aim at providing a Wikipedia-based solution for a question answering system. Wikipedia itself is not very formalized, but previous research shows that it can be formalized.[20,21] DBpedia[22] and YAGO[23]-based solutions are viable for classes 1 and 2 questions. Other researchers also follow this route. Adel Tahri *et al.* proposed a Support Vector Machines-based algorithm for a DBpedia-based question answering system.[24] QASYO[25] is a YAGO-based system designed to use YAGO ontology to answer questions. Mohamed Yahya *et al.* combined DBpedia and YAGO as sources to their approach and generated SPARQL queries based on the questions asked.[26]

3. DBpedia and YAGO as Knowledge Bases

DBpedia and YAGO are based on the Wikipedia project. The system itself has a very simple, and rather poor structure as it is dedicated for human readers. Such readers

need a proper and consistent graphical interface to be able to perform information lookup. The internal structure is of secondary concern. For automated systems the approach has to be completely different. The internal structure is what matters, the presentation can be thus completely ignored. The DBpedia and YAGO projects aim at extending Wikipedia functionality in this area.

The DBpedia introduced an ontology consisting of 492 classes with 53827 predicates. The ontology contains some well-defined concepts like city, music album, language, country, etc. Entries of the same type show a high level of similarity. In some cases, however only domain-specific facts are available, e.g., human beings only provide information like name, date of birth and nationality for all of the concepts. The drawback of the ontology is also the fact that most of the properties were derived from infoboxes only. Those however differ even for the same type of objects. DBpedia also lacks in internal interlinking. It uses the same grouping mode as Wikipedia, namely, aggregation based on Wikipedia categories.

YAGO, contrary to DBpedia, is developed by a limited number of dedicated experts, supervised by Max Planck Institute in Saarbrcken. The project goal is to aim for accuracy rather than the size of the gathered resources. As such YAGO is far more formalized than DBpedia. It also uses WordNet taxonomy to enrich the concept structure. The accuracy of YAGO datasets is very high, reports show that it is reaching 95%.[23]

With extensions introduced by YAGO and DBpedia, the Wikipedia can be treated as a viable source of knowledge. Even though the formalization is still lacking and no all concepts are properly described the quality of both solutions is constantly improving. The two sets are even connected with each other providing means of interoperability between both solutions.

4. Using Fibers in Answer Lookup

Ontologies both YAGO and DBpedia provide SPARQL endpoints for querying their data.[a,b] The aim of our work is to convert user input, namely, classes 1 and 2 questions formulated using natural language, into such queries. The approach consists of three phases (Fig. 1).[27]

The first stage, the query analysis, performs its grammatical parsing. During this stage a Stanford NLP Parser[28] is used to detect the structure of the query and convert it into an ordered tree.[29] When the query is converted into a tree structure, the expected relations in the question are denoted as a set of parallel branches of the same subtree with distance from each other equal to one. Example of such tree for question *What is height of Eiffel Tower?* is presented in Fig. 2.

The next step of this stage is concept retrieval from a parsed query sentence. This step is crucial as in the next stages only information for these concepts will be

[a] https://linkeddata1.calcul.u-psud.fr/sparql.
[b] https://dbpedia.org/sparql.

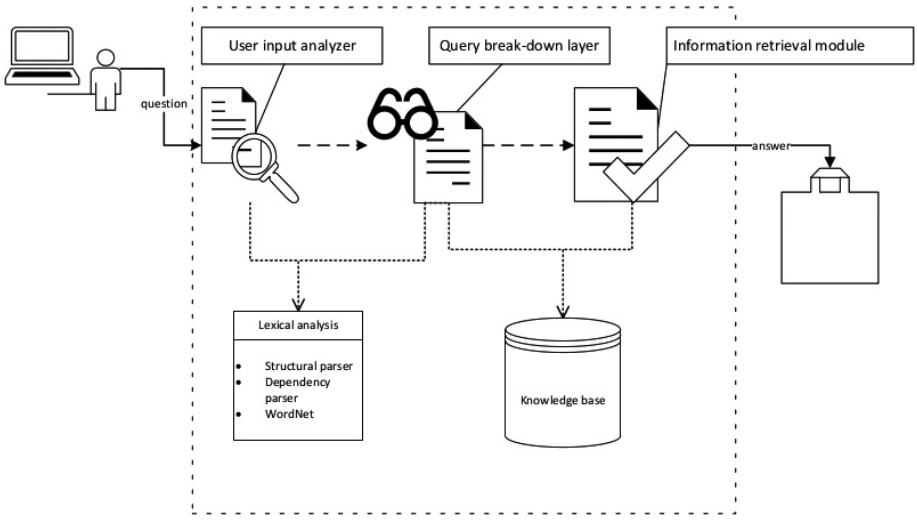

Fig. 1. Query answering process.

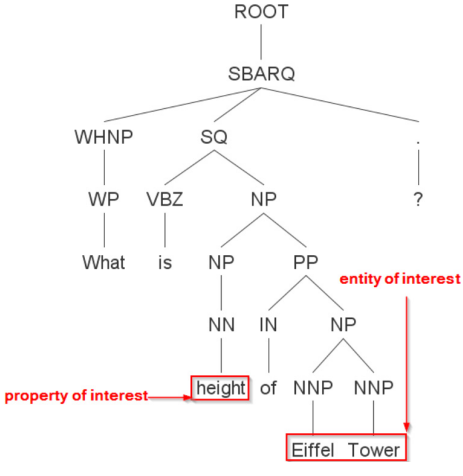

Fig. 2. Example results of Stanford NLP Parser.

retrieved. To find every known concept the aforementioned tree is traversed using Algorithm 1.

where

- `flattenTree()` — is building string representation of tree leaves, effectively retrieving chunk of sentence built from logically connected words,
- `isStopWord()` — checks whether entity candidate is not one of the words commonly known as usual in language, which in general dont provide any value in terms of concept retrieval,

```
def annotateEntities(parsedTree):
  possibleEntity = flattenTree(parsedTree.root)
  if not isStopWord(possibleEntity)
    if knowledgeBase.isKnownConcept(possibleEntity)
      knownEntities += possibleEntity
      entityFound = True

  if not entityFound
    for each child in parsedTree.root
      annotateEntities(subtree(child))
```

Listing 1. Finding every known concept in dependency tree.

- `isKnownConcept()` — checks whether there exists any concept in the database which textual representation (label) is similar to the provided name; candidate entries are retrieved and then ranked with similarity score; only concepts with high enough score are assigned as candidates for concept under question.

In the example, the goal is to recognize both height property and Eiffel Tower entity. At this step, the databases are queried using SPARQL with concepts detected during the previous stage, where concepts are detected in the query — usually the entities and their properties. The user query is tagged using Penn–Treebank notation and is supplied with a list of detected entities and their properties.

The output from the first stage provides basic information on the user's query.

The second stage, the query break-down, aims at determining the entity and properties in the query. The first step of this stage is dependency parsing. The grammatically parsed query is broken down into basic semantically connected concepts. The dependency parser logically binds the concepts and determines the relation between them creating a dependency graph. The graph is conceptually close to grammatical sentence structure but it omits syntactical alterations not relevant to the meaning of the query. As an example can be the question *Who is author of Game of Thrones?*. The raw output from dependency parser is as follows: cop(Who, is), nsubj(Who, author), nmod(Game, author), case(Game, of), nmod(Game, Thrones) and case(Thrones, of).

In the next step of this stage, the output of the dependency parser is minimized, as only meaningful relations found by the dependency parser will be taken into further consideration (Algorithm 2).

where

- `stageOneResult` — results of grammatical parsing and concept retrieval encapsulation in helper class, which allows their exploration,
- `dependencies` — raw list of dependencies retrieved from the parser,
- `isQuestionRoot()` — function determining whether this is first meaningful node of question (usually Wh-word),

```
def retrieveRelations(stageOneResult, dependencies):
  relations = []
  for each dependency in dependencies:
    for word in dependency.words:
      if stageOneResult.isQuestionRoot(word):
        word.setQuestionRoot()
      if stageOneResult.isKnownConcept(word):
        word.setKnownConcept()

    if not stageOneResult.theSameConcept(dependency.words):
      relations += Relation(dependency)

  return relations
```

Listing 2. Minimization of dependency parser output.

- `isKnownConcept()` — checks if given word is part of named entity recognized in previous phase,
- tt theSameConcept() — used to determine whether detected dependency is connecting two parts of the same named entity.

Then the set produced by our algorithm is converted into a tree. We assume that the question will have only one root determined during relation processing. Starting from the root, relations are bind together to form a dependency tree. Then the dependency tree is analyzed. We observed that dependency trees of most of the queries had one or more paths leading from the root to named entity of interest. These paths (referenced further as fibers) are candidates for the representation of actual user intention. The goal of this step is to retrieve minimal fiber (marked with red color in Fig. 3).

The output of the query break-down stage, namely, the dependency parsing combined with data retrieved in the previous step, effectively produces final input for the retrieval subsystem. This data consists of fibers that are minimized to contain

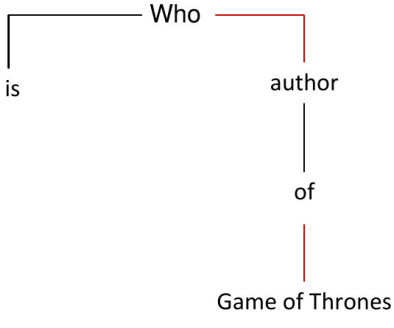

Fig. 3. The minimized dependency tree for question *Who is author of Game of Thrones?*.

```
def findAnswer(fiber):
  answer = None
  leaf = fiber.leaf
  relation = leaf.parent
  leafPredicates = knowledgeBase.getAllPredicates(leaf)

  kbRelation = leafPredicates.findMostSimilar(relation)
  if kbRelation not None:
    newLeaf = knowledgeBase.resolve(leaf, kbRelation)
    if newLeaf.parent == Root:
      answer = newLeaf
    else:
      fiber.remove(leaf)
      fiber.remove(relation)
      fiber.append(newLeaf)

      answer = findAnswer(fiber)

  return answer
```

Listing 3. Information retrieval.

only meaningful data required for retrieving triples from the knowledge base. The fibers are ranked based on their length and number of known concepts, the one with the shortest path leading to concepts in question is the most probable candidate for actual user intention representation.

During the last stage, the information retrieval, the fibers are sorted based on their score of relevancy and evaluated. The first answer retrieved from the database (based on the fiber converted into SPARQL query) is presented to the user. The queries are based on concepts retrieved from the databases at previous stages of query analysis. The whole procedure is formally presented in Algorithm 3.

where

- `knowledgeBase` — wrapper class performing SPARQL queries on underlying SPARQL endpoint,
- `getAllPredicates()` — method returning number of relations connected with specified concept,
- `findMostSimilar()` — heuristic method utilizing WordNet distance and string similarity methods to retrieve the most accurate candidate for required relation,
- `resolve()` — retrieves result of relation from knowledge base.

If the question is of class 2, the above process has to be performed for each detected subquestion, like presented in Fig. 4.

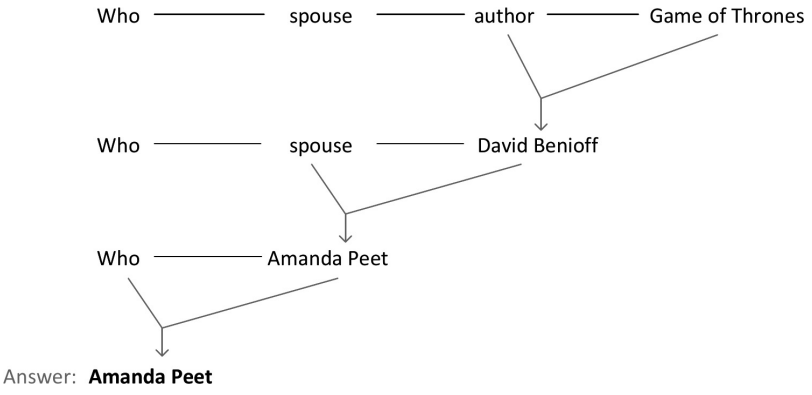

Fig. 4. Information retrieval for class 2 question.

Table 1. Answer correctness and performance scores.

True positives	True negatives	False positives	False negatives	Recall	Precision	Specificity	Accuracy	F-Measure
8	18	4	14	0, 36	0, 67	0, 82	0, 59	0, 47

The approach was tested using a subset of TREC-8[c] questions that were of classes 1 and 2. For each question we analyzed the results based on the following criteria: is the looked-up information available in the database, was the correct answer retrieved (true positive), the approach presented incorrect answer (false positive). The summary of the results is presented in Table 1.

After a closer look at many of the results that were unanswered or wrongly answered (and had correct answers within the databases), we decided to extend our solution.

5. Extended Fiber-Based Solution

Both the original and extended solutions are looking for answers to the queries based on paths in dependency trees representing the user's query. The original implementation, presented in Sec. 4, had several drawbacks limiting its quality. We identified and tried to eliminate these drawbacks which allowed us to improve the quality of the proposed methods. The solution allows one to submit queries and retrieve answers along with a full description of the analysis process. We are not focused on the performance of the algorithms as we use online YAGO and DBpedia endpoints which significantly influence the performance.

The first problem we identified in the original solution was the method used to match question elements to attribute names. The original method used the

[c] http://trec.nist.gov/data/qa/T8_QAdata/topics.qa_questions.txt.

Levenhstein[30] measure which gave high similarity for words with matching parts. Sequences like "death place" were thus matched to attributes like "deathPlace" and "birthPlace". This resulted in additional, usually wrong answers, e.g. for question *What is death place of Mohammad Khaled Hossain?* it produced six answers: *Mount Everest*, *Nepal*, *2013-05-21*, *2013-5-20*, *Bangladesh*, *Munshiganj*. We decided to use the difflib Python module[d] for the matching which allowed us to eliminate the additional answers. After changes, the system retrieves two, proper answers: *Mount Everest*, *Nepal*.

We also extended the algorithm for attributes matching. In the original solution the parts of the query were matched to the label of the attributes found in the databases used. After analysis, it occurred that the actual name of the attribute is often almost a direct match (whereas the label can be misleading).

Further, we also included redirection in possible matches for query attributes. If the attribute could not be matched but low-scored candidates included redirection we followed those. This way it is possible to match entities like alternative names of cities or countries (e.g., *Ulan Bator* and *Ulanbaatar*).

The original solution did not analyze synonyms when looking for entities or attributes matches. In the current implementation, we use WordNet[31] as a backend for the nltk Python module[e32] for extending the list of attributes checked. For example the original program could not answer the question *Who is the partner of Donald Trump?* as the databases do not have the attribute *partner* to identify the proper entity. The database contains however an attribute *spouse*. Using synonyms we can modify the question into *Who is a spouse of Donald Trump?* and thus get the correct answer. We also use the same mechanism when an attribute or its synonym cannot be found in the databases. In this case, we use the nltk module to find the closest words for missing attributes and use them to generate the answer. An example of a question requiring such actions is *What is decease place of Chopin?* as neither *decease place* nor its synonyms could be found in the databases for the entity *Chopin*. This way, we are able to generate alternative questions that should have the same answer, and in this way reach the answer.

Attributes name matching sometimes gave misleading results, e.g., in the question *Where is birth place of Jimi Hendrix?* attribute *birth place*, due to different similarity measures used,[33] was mapped to attributes *birthPlace* and *birthDate*. To bypass the limitation of similarity measures in our approach, we try to detect interrogative pronouns within the question and their domain. For answer generation, we analyze only those matched attributes that share the same domain as detected from the interrogative pronoun of the question. This increased the quality of the answers but introduced another problem. In some cases, the attribute is not within the same domain as the interrogative pronoun but the answer is, e.g., in the question *Who is a successor of George Washington?* the attribute *successor* is not the attribute of a

[d]https://docs.python.org/2/library/difflib.html.
[e]https://www.nltk.org/.

person. However, the answer to the question, like *John Adams*, is. Thus, after finding a potential answer the domain test must be performed to check whether the answer is from the same domain regardless of the attribute that led to the answer.

The same nltk module also allowed us to ask questions where entities' attributes are represented as actions, e.g., *When died Jim Morrison?*. Based on the verbs in the question we generate nouns, which are in turn the most often used as attributes names within DBpedia and YAGO. In this case, there is no *Jim Morrison* entity attribute called *died*. There is, however, a proper answer stored under *deathDate*, which, combined with the domain deduction described earlier is derived from noun *death*, which in turn was generated using the nltk module from the verb *died*. So the answer was 1971-07-03.

The biggest problem occurred for entities with multiple meanings. Many words or names in natural language can have more than one meaning, e.g., entity *Washington*[f] which can have a high number of pages related to. Such entities are usually represented using disambiguation entities that have references to different meanings of the entity in question. We look through those referenced entities for a potential answer and then match the domain of the referenced entity, the attribute and the interrogative pronoun to present the best answer. This mechanism still needs further work as it generates additional, mostly wrong, answers. The list of alternative meanings can also be long.

6. Evaluation of the Extended Solution

We evaluated the extended solution using the aforementioned TREC-8 questions that belonged to classes 1 and 2. Out of the 40 questions, 32 had answers in the DBpedia and YAGO databases, the remaining 8 did not. For all non-answerable questions the algorithm did not give any answers. Out of 32 answerable questions the algorithm answered correctly 20 questions. In 2 cases, we got wrong answers and the remaining 10 questions were left unanswered. In 7 cases, the algorithm gave additional, wrong answers. Those additional answers are the reason why the sum of true/false positives/negatives is higher than the number of questions used for the evaluation.

The results of correctness and achieved performance evaluated using precision, recall and F-measure are presented in Tables 2 and 3, respectively.

The updated algorithm achieved much higher recall (0.625 vs. 0.36 in our evaluation implementation) while keeping precision at the same level (0.69 vs. 0.67). The final F-measure increased from 0.47 to 0.66.

Table 2. Answer correctness.

True positives	True negatives	False positives	False negatives
20	8	9	12

[f]https://en.wikipedia.org/wiki/Washington.

Table 3. Performance scores.

Precision	Recall	F-Measure
0.69	0.625	0.66

Table 4. Examples of questions with wrong answers.

Question	Answer in database	Correct answer returned	Wrong answer returned	Comment
What is location of US Declaration of Independence?	yes	no	yes	"position" from "location" and "position" property holds a value for picture ("right")
Who is leading actor in "The Godfather"?	yes	no	yes	"direct actor" reasoned as synonym to "leading actor", then associated with property "director"
What is date of Battle of the Somme?	yes	yes	yes	wrong answers got from property "seeAlso" ("see" was obtained as a synonym to word "date")
What is location of Taj Mahal?	yes	yes	yes	wrong answers got from property "caption" (which has 80% of similarity with word "location")
What is population of Tucson?	yes	yes	yes	wrong answers from properties "populationAsOf", "populationMetro", "populationUrban", "populationBlank", "populationEst" (too similar to "population")
What is population of Ulan Bator?	yes	yes	yes	wrong answer from property "populationAsOf" (too similar to "population")
What is population of Ushuaia?	yes	yes	yes	wrong answers from property "populationAsOf" (too similar to "population")
What is produced by Peugeot?	yes	yes	yes	returned also answer "1739000" from property "production"
Where was Washington born?	yes	yes	yes	problems with disambiguation

As can be seen in Table 4, there are still issues with some types of questions. In some cases, the attributes detected within the query were matched incorrectly to entity attributes in the databases. In some cases, this leads to retrieving completely wrong answers. It can be observed, e.g., in question *Who is a leading actor in "The Godfather"?* where the attribute *leading actor* was marked as a synonym to *direct actor* and as a result matched to the attribute *director*. In other cases, the wrong match generated additional answers, like in question *What is the location of Taj Mahal?* where word *location* had borderline similarity value of 80% with attribute *caption*. The same error can be observed in the question *What is the population of Tucson?*. In this case, the word *population* was incorrectly matched to attributes

populationAsOf, populationMetro, populationUrban, populationBlank and *populationEst*. Such results were observed in almost all questions with additional, usually wrong, answers. In almost all cases however, the algorithm was able to give the correct answer alongside the wrong ones. This proves that the query analysis and entity detection process is correct. In further works, we will focus on algorithms for better attribute matching.

In one case (*Who is Voyager manager?*) the question is formulated in a way that, even for humans, it is hard to determine the correct answer without additional context information. We might assume the question asks for the manager of the Voyager space program, however, the algorithm decided to match the ambiguous name *Voyager* to a TV show *Earth Star Voyager*[g] and gave as the answer the name of the show director (which from a technical point of view can be treated as correct). In this case, the algorithm should be able to generate more potential answers from different domains. The system could present those answers and ask the user the question that would allow establishing the entity domain. We test our algorithm for narrowing the domain using interaction with the user[34] and if the future we plan to adapt it to our system.

7. Conclusions

The proposed solution allows, in most cases, answering the questions formulated in natural language and is not domain-specific. The strength of the proposed idea lies also in the usage of widely available tools and databases. In most cases, such a system is also able to give a direct answer to the question asked (e.g., the date of an event in question). With certain improvements, mainly in matching and disambiguation algorithms, the system might be able to answer questions belonging to class 3 type of questions (answering based on multiple sources). The proposed query analysis method may also be used to extract semantic data from text.

The advantage of the system is its speed and small hardware requirements. The system can be run on a standard PC and does not require any other special resources. Many other solutions (especially IBM Watson) requires dedicated and powerful hardware.

Most of the problems found with the method are related to insufficient quality of attributes matching. Such cases were observed during analysis of results given for almost all questions with additional, usually wrong, answers. The algorithm however, in most cases, was able to give the correct answer alongside the wrong ones. This proves that the query analysis and entity detection process is correct.

Most of the wrong answers can also be eliminated by formalization and harmonization of attributes used to describe concepts within the same and different domains in the databases itself. The DBpedia and YAGO databases still use very shallow and not well interlinked internal ontologies. Data linking occurs on different

[g] https://en.wikipedia.org/wiki/Earth_Star_Voyager.

levels and is very domain-dependent. Constant development and formalization of the databases used can thus have a positive impact on the results obtained by the algorithm, and in time should have a positive impact on the quality of the proposed method.

The problems described above involves a difficult problem which is natural language disambiguation. Further improvements of the proposed solution should be thus focused on improving the matching algorithm and the ability to better understand the context and domain of the questions asked. We believe, that after further improvements, the proposed method can be used as a base for a system capable to answer questions formulated in natural language.

Acknowledgments

The paper is an extended version of a paper presented during the 10th International Conference on Computational Collective Intelligence (ICCCI 2018) that held between 5th and 7th September 2018 in Bristol, UK and can be found in Boiński T., Szymański J., Dudek B., Zalewski P., Dompke S., Czarnecka M. (2018). DBpedia and YAGO Based System for Answering Questions in Natural Language, 11055 (Chap. 35), pp. 383–392, https://doi.org/10.1007/978-3-319-98443-8_35.

This paper introduces the following changes: In Sec. 3, the lexical databases were described and evaluated as knowledge bases suitable for natural language question answering. Section 4 also contains a detailed description of the algorithm and its preliminary evaluation. The title was changed to better reflect the contents of the paper.

References

1. R. F. Simmons, Natural language question-answering systems: 1969, *Commun. ACM* **13**(1) (1970) 15–30.
2. J. Szymański, Words context analysis for improvement of information retrieval, in *Computational Collective Intelligence. Technologies and Applications* (Springer, 2012), pp. 318–325.
3. W. Duch, J. Szymański and T. Sarnatowicz, Concept description vectors and the 20 question game, in *Intelligent Information Processing and Web Mining* (Springer, 2005), pp. 41–50.
4. D. I. Moldovan, S. M. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, R. Girju and V. Rus, Lasso: A tool for surfing the answer net., in *Text REtrieval Conf.*, Vol. 8 (Gaithersburg, Maryland, USA, 1999), pp. 65–73.
5. A. Bouziane, D. Bouchiha, N. Doumi and M. Malki, Question answering systems: survey and trends, *Procedia Comput. Sci.* **73** (2015) 366–375.
6. A. Mishra and S. K. Jain, A survey on question answering systems with classification, *J. King Saud Univ. Comput. Inf. Sci.* **28**(3) (2016) 345–361.
7. A.-M. Popescu, O. Etzioni and H. Kautz, Towards a theory of natural language interfaces to databases, in *Proc. 8th Int. Conf. Intelligent user interfaces* (ACM, 2003), pp. 149–157.
8. B. F. Green Jr, A. K. Wolf, C. Chomsky and K. Laughery, Baseball: An automatic question-answerer, in *Papers Presented at the May 9–11, 1961, Western Joint IRE-AIEE-ACM Computer Conf.* (ACM, 1961), pp. 219–224.

9. Trec, *Text REtrieval Conf.* (TREC), http://trec.nist.gov/ [Online: 12.03.2018].
10. TREC, Text REtrieval Conf., 2008, http://trec.nist.gov/data/qa/T8_QAdata/topics.qa_questions.txt [Online: 19.11.2017].
11. D. Laurent, P. Séguéla and S. Nègre, Cross lingual question answering using qristal for clef 2006, in *Workshop of the Cross-Language Evaluation Forum for European Languages* (Springer, 2006), pp. 339–350.
12. O. Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, L. Monceaux, I. Robba and A. Vilnat, Finding an answer based on the recognition of the question focus., in *Text REtrieval Conf.* (Gaithersburg, Maryland, USA, 2001), pp. 362–370.
13. S. M. Harabagiu, D. I. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. C. Bunescu, R. Girju, V. Rus and P. Morarescu, FALCON: Boosting Knowledge for Answer Engines, in *Text REtrieval Conf.* (Gaithersburg, Maryland, USA, 2000), pp. 479–488.
14. M. Banko, E. Brill, S. Dumais and J. Lin, AskMSR: Question answering using the worldwide Web, in *Proc. 2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases* (Palo Alto, CA. Menlo Park, CA, 2002), pp. 7–9.
15. S. Ferré, Squall: A controlled natural language as expressive as sparql 1.1, in *Int. Conf. Application of Natural Language to Information Systems* (Springer, 2013), pp. 114–125.
16. A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann and D. Gerber, Sorry, I don't speak SPARQL: Translating SPARQL queries into natural language, in *Proc. 22nd Int. Conf. World Wide Web* (ACM, 2013), pp. 977–988.
17. P. Cimiano, P. Haase and J. Heizmann, Porting natural language interfaces between domains: An experimental user study with the orakel system, in *Proc. 12th Int. Conf. Intelligent User Interfaces* (ACM, 2007), pp. 180–189.
18. D. A. Ferrucci, IBM's Watson/DeepQA, in *SIGARCH Computer Architecture News*, Vol. 39 (ACM, New York, NY, USA, 2011). https://dl.acm.org/doi/10.1145/2024723.2019525.
19. D. Ferrucci, A. Levas, S. Bagchi, D. Gondek and E. T. Mueller, Watson: Beyond jeopardy!, *Artific. Intell.* **199** (2013) 93–105.
20. J. Szymański, Self–organizing map representation for clustering wikipedia search results, in *Asian Conf. Intelligent Information and Database Systems* (Springer, 2011), pp. 140–149.
21. J. Szymański and W. Duch, Semantic memory knowledge acquisition through active dialogues, in *Int. Joint Conf. Neural Networks IJCNN 2007* (2007, IEEE), pp. 536–541.
22. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, Dbpedia: A nucleus for a web of open data, in *The Semantic Web* (Springer, 2007), pp. 722–735.
23. F. M. Suchanek, G. Kasneci and G. Weikum, Yago: A core of semantic knowledge, in *Proc. 16th Int. Conf. World Wide Web* (ACM, 2007), pp. 697–706.
24. A. Tahri and O. Tibermacine, Dbpedia based factoid question answering system, *Int. J. Web Semantic Technol.* **4**(3) (2013) 23.
25. A. M. Moussa and R. F. Abdel-Kader, Qasyo: A question answering system for YAGO ontology, *Int. J. Database Theory Appl.* **4**(2) (2011) 99–112.
26. M. Yahya, K. Berberich, S. Elbassuoni and G. Weikum, Robust question answering over the web of linked data, in *Proc. 22nd ACM Int. Conf. Information Knowledge Management* (ACM, 2013), pp. 1107–1116.
27. T. Boiński and A. Ambrożewicz, DBpedia and YAGO as knowledge base for natural language based question answering the evaluation, in *Int. Conf. Man-Machine Interactions* (Springer, 2017), pp. 251–260.
28. M.-C. De Marneffe, B. MacCartney, C. D. Manning *et al.*, Generating typed dependency parses from phrase structure parses, *Proc. LREC*, **6** (2006) 449–454.

29. M. P. Marcus, M. A. Marcinkiewicz and B. Santorini, Building a large annotated corpus of English: The Penn Treebank, *Comput. Linguistics* **19**(2) (1993) 313–330.
30. V. I. Lcvenshtcin, Binary codes capable of correcting deletions, insertions, and reversals, in *Cybernetics and Control Theory*, Vol. 10 (Soviet Physics-Doklady, 1966), pp. 707–710.
31. G. A. Miller, R. Beckitch, C. Fellbaum, D. Gross and K. Miller, *Introduction to WordNet: An On-line Lexical Database* (Cognitive Science Laboratory, Princeton University Press, 1993).
32. S. Bird, E. Klein and E. Loper, *Natural Language Processing with Python* (O'Reilly Media, 2009).
33. P. Czarnul, J. Kuchta, M. Matuszek, P. Rościszewski, J. Proficz, M. Wojcik and J. Szymański, MERPSYS: An environment for simulation of parallel application execution on large scale HPC systems, *Simul. Model. Practice Theory* **77** (2017) 124–140.
34. J. Szymański and W. Duch, Information retrieval with semantic memory model, *Cognitive Syst. Research* **14**(1) (2012) 84–100.