A Cooking Knowledge Graph and Benchmark for Question Answering Evaluation in Lifelong Learning Scenarios

Mathilde Veron, Anselmo Peã±as, Guillermo Echegoyen, Somnath Banerjee, Sahar Ghannay, Sophie Rosset

A Cooking Knowledge Graph and Benchmark for Question Answering Evaluation in Lifelong Learning scenarios

Mathilde Veron[1,3], Anselmo Peñas[2], Guillermo Echegoyen[2], Somnath Banerjee[1], Sahar Ghannay[1], and Sophie Rosset[1]

[1] LIMSI, CNRS, Université Paris-Saclay, Orsay, France
firstname.lastname@limsi.fr
[2] UNED NLP & IR Group, Madrid, Spain {anselmo,gblanco}@lsi.uned.es
[3] LNE, Trappes, France

Abstract. In a long term exploitation environment, a Question Answering (QA) system should maintain or even improve its performance over time, trying to overcome the lacks made evident through the interactions with users. We claim that, in order to make progress in the QA over Knowledge Bases (KBs) research field, we must deal with two problems at the same time: the translation of Natural Language (NL) questions into formal queries, and the detection of missing knowledge that impact the way a question is answered. The research on these two challenges has not been addressed jointly until now, what motivates the main goals of this work: (i) the definition of the problem and (ii) the development of a methodology to create the evaluation resources needed to address this challenge.

Keywords: Question Answering · Lifelong Learning · Evaluation resources.

1 Introduction

Since every human domain is dynamic and evolves over time, in the mid-long term, any Knowledge Base (KB) will become incomplete or, at least, it won't we able to satisfy user demands of information. In a long term exploitation environment, QA systems must deal with the challenge of maintaining their performance over time and try to overcome the lacks made evident through the interactions with the users. In other words, we need to provide QA systems with Lifelong Learning mechanisms. The first step is the detection of such situations. QA systems must distinguish the reason why the system cannot answer a question: either the problem is in the translation of the Natural Language (NL) question into a formal query, or the problem is a lack of knowledge that prevent the system from giving an answer. If the reason is the latter, the system must trigger a learning process to overcome this limitation and update its previous knowledge.

Unfortunately, most of the current research in QA over KBs works with datasets of questions that can always be answered by the KB [7,4]. That is to

say, the research focus is on the problem of how to translate a NL question into a formal query. However, working under the assumption that there exists an answer to the question according to the KB leads researchers to a set of solutions that will not work in a real scenario.

We claim that, in order to make progress in the QA research field, we must deal with both problems at the same time: the translation of NL into formal queries, and the detection of lacks of knowledge that impact the way questions are answered. To the best of our knowledge, the effort done in this direction has not been significant. Therefore, the main goals of this work are (i) the definition of the problem and (ii) the development of a methodology to create the evaluation resources needed to address this challenge.

For a better understanding of the problem, we chose the context of a real user demand, constructing an imperfect (by definition) Knowledge Graph (KG), and asking real users to pose questions that the QA system has to answer. Then, a set of annotators have tried to translate the real NL questions into formal queries, identifying when the questions can be translated and when they cannot, annotating the reasons why. Examples of annotations can be found in Figure 1.

The form of the Knowledge Base is a Graph (i.e. RDF triples style) for several reasons. First, the updating of the KG with new classes (or types), property names (or relations), instances (or objects), etc. is straightforward and does not affect the previous version. It only requires the addition of new triples. Secondly, working with a graph makes the use of different formalisms and different retrieval engines possible, from using SPARQL over database managers (like Virtuoso) to the use of simple Prolog. That is, in a Lifelong scenario where the systems must evolve over time and continuously update their knowledge, KGs seem to be the most appropriate formalism.

In the following sections, we describe the whole process in detail, together with our learnings and conclusions. The contributions of this work are:

- The definition of the problem;
- A methodology for studying it and creating the evaluation resources;
- A publicly available Knowledge Graph (in cooking domain)[4];
- A first version of a set of answerable and unanswerable questions over this KG, for benchmarking system self-diagnostic about the reasons why the question cannot be answered by the KG[5].

2 Previous work

2.1 Question Answering with unanswerable questions

We are interested in QA systems with the ability to recognize unanswerable questions. This problem has been addressed lately under the free text assumption and only partially. To the best of our knowledge, it has never been addressed in QA over KBs.

[4] http://nlp.uned.es/lihlith-project/cook/

[5] https://perso.limsi.fr/rosset/resources/cooking_LL_QA.json

Under the free text paradigm, systems must answer questions whose answer can be found in a given text. Current research is more focused on answer extraction than in the complete QA architecture that includes the recovering or ranking of candidate paragraphs. (as opposed to KG-based QA, where the whole process must be carried out). SQuAD [12], TriviaQA [5] and MS Marco [11] are among the most popular collections for QA over free text featuring empty answers. They are all created following one or various crowdsourcing annotation processes. Current systems competing with these datasets are usually made out of ensembles of pre-trained language models like ALBERT [6] and XLNet [13].

However, when doing QA over KBs, a more sophisticated process is required. In general, all systems proceed with a multi-step process, comprising a combination of complex steps: Question Analysis, Named Entity Recognition and Linking, Disambiguation and Parsing. There are some surveys detailing these systems, we refer the reader to them [2], and [4]. Over the last years, neural systems have tremendously increased in capability, however in the specific domain of QA over KBs, it has been argued that deep learning does not contribute that much [10]. In particular, these systems can, for now, only answer simple questions [1,3,8]. Furthermore, to solve QA over KBs, the majority of approaches assume that the question can be answered by the KG because the most popular collections like QALD [7] or LC-QUAD [4] do not contain empty answers. Therefore, answering a question is a kind of graph matching against the KG.

In summary, a production system for QA over KBs requires the ability to recognize unanswerable questions, and therefore, we identify the need to correctly define the problem of QA over KBs, but also to develop the necessary resources to train and evaluate systems to solve this problem.

2.2 Lifelong learning and Question Answering

This problem has already captured the attention of some researchers such as Mazumder and his colleagues [9] although in that work, the problem is only addressed partially. In particular, queries to the system are just single triples, reducing to the trivial case the problem of deciding whether the answer to a question is in the KG or not. It simplifies also the problem of detecting the pieces of knowledge that have to be added to the KG. The option taken for enriching the KG is to ask the user for some missing pieces of knowledge and try to find strategies to infer some others. However, in the general scenario of complex NL QA over KGs these decisions are not trivial. If a system does not get an answer to a question, it could be due to several factors, including some errors in the process of NL interpretation (e.g. Entity Linking).

3 Cooking Knowledge Graph construction

The KG is a set of triples `<arg1, property-name, arg2>`, where the first argument must be always an entity and the second one can be both an entity or a literal (number or string). Entities (also mentioned as resources or objects)

can refer to type names (or classes, e.g. `cookbook:ingredients`), instances (e.g. `cookbook:milk`), or category names (e.g. `category:pancake_recipes`). Categories refer to groups of recipes according to some criteria given by the original wiki. Thus, a recipe can belong to several categories and this will be encoded through the corresponding triples with the property name `recipeCategory`. Recipe categories use the prefix `category:` instead of `cookbook:` used for the rest of entities. The property names (or relations) used here follow the Recipe schema[6] when it has been possible. The complete set of properties is shown in Table 1.

The KG has been derived from the English wikibook (enwikibooks-20190701-pages-articles.xml) related to cooking (name space 102, Cookbook). We have processed both the cookbook pages one by one, and the category links file (enwikibooks-20190701-categorylinks.sql).

The processing of the category links file produced 480 triples among categories, 6935 triples that link recipes to recipe categories, and 4479 type relationships.

With respect to the processing of the Cookbook enwikibook pages, the method has been the following:

1. Consider the mark *redirect* inside the wiki pages to store which is the canonical *uri* for each object.
2. Use the *title* of the page to produce the *name* and *url* relations.
3. Identify sections in recipe pages.
4. Process ingredients section and produce the triples for the *recipeIngredient* and *recipeFoodstuff* relations. The former one corresponds to the text describing the ingredient, quantity, etc. The latter one, corresponds to the object in the Knowledge Graph associated to the food used as ingredient.
5. Process instructions section to produce a triple that relates the recipe with the list of steps (*recipeInstructions* relationship). Each element in the list corresponds to the original text describing the step.
6. Process the section of notes and variations, and produce the triple for the *recipeNotes* between the recipe object and the corresponding text.
7. Process the marks that associate the main types with the page under processing. These main types are: recipes, diets, equipment, ingredients and techniques. Main ingredient subtypes are: *cereals, chesses, fruits, herbs_and_spices, meat_and_poultry, nuts_and_seeds, seafood* and *vegetables*. These types and subtypes are the ones defined originally in the wikibook. We observed that, in many cases, there are inconsistencies in the types of the objects. In principle, since the types described are a partition, no objected should have more than one of these main types. However, in the majority of cases it is the type recipes the one that spreads incorrectly over the objects. In these cases, we promote the other types.
8. Process the links in the page with two purposes: (i) to store their text as label for the corresponding objects and (ii) identify that there is an implicit relationship between two objects. In the wiki pages there exist a number of

[6] `https://schema.org/Recipe`

links that relate the current page to another page. In our case, both pages will correspond to objects in the knowledge graph, but we need to infer the appropriate property name that must identify this relationship. Once the types of the source and target objects are solved according to the previous step, then the implicit relationship between them is trivially solved and made explicit.

9. Process the recipe *summary* according to the corresponding template instructions. This processing produces the triples for *recipeCategory*, *recipeYield* (servings), *totalTime* and *difficulty* (numeric value from 1 to 5).

	Freq.	Property name	Example
General props	8263	label	baguette **label** "french bread"
	5214	url	adobo **url** en.wikibooks.org/wiki/Cookbook:Adobo
	5214	name	frosting_and_icing_recipes **name** "Frosting"
	5156	type	baking_soda **type** cookbook:leavening_agents
Recipe properties	21077	recipeIngredient	chocolate_mousse **recipeIngredient** "200 g bitter..."
	15616	recipeCategory	chocolate_mousse **recipeCategory** category:dessert_recipes
	12343	recipeFoodstuff	chocolate_mousse **recipeFoodstuff** cookbook:chocolate
	2419	recipeInstructions	chocolate_mousse **recipeInstructions** ["Melt chocolate..."]
	849	difficulty	chocolate_mousse **difficulty** 2
	844	totalTime	chocolate_mousse **totalTime** "30 minutes"
	805	recipeYield	chocolate_mousse **recipeYield** 4
	458	recipeNotes	chocolate_mousse **recipeNotes** "* This recipe is not the..."

Table 1. Property names in the Cooking KG

4 Methodology for dataset creation

This section describes the creation process of the developed dataset. The objective of this dataset is to help the research community to study the following research issues: (i) the translation of NL into formal queries, (ii) the detection of unanswerable questions and (iii) the identification of elements missing in the KG which impact the way questions are answered. We first describe how we collected the user's queries in NL and then how we annotated them.

4.1 Collection of queries in natural language

We asked collaborators from our institutions through a web form to write at least 5 queries in natural language in English. The participants were no native English speakers but Spanish and French people. We received 30 responses in 3 days, resulting in 169 queries. The participants needed around 5 minutes to read the guidelines and write at least 5 queries. They were asked to pose any question about the cooking domain. Thus we provided them a non exhaustive list of items they could ask about along with some examples. The query could be posed in any of these four possible ways: interrogative (e.g., "Which herbs go well with mushrooms?"), imperative (e.g., "Give me a soup recipe for tonight"),

informative ("I'm looking for the name of the utensil that is used to beat the egg whites"), or propositional (yes/no question e.g. "Is tomato a fruit?") and have to fit in only one sentence.

After collecting the queries in NL, we filtered them by assessing their usability regarding our task[7]. It allowed us to directly discard the queries that couldn't be answered either with the current KG or by adding new elements to the KG. Each question has been annotated as usable or not by two different persons. After filtering, 124 queries were identified as usable (around 73%).

4.2 Annotations

ID	User's query in NL	Prolog query	Result	Reasons why the query cannot be answered		
				type of the missing element		missing element
1	When is the apple's season?	r(cookbook:apple, type, Season_ref), r(Season_ref, type, cookbook:ingredients_by_seasonality).	Season_ref = cookbook:autumn_ingredients; Season_ref = cookbook:summer_ingredients	entity	☐	
				property name	☐	
				type name	☐	
				triple	☐	
2	Can I use goat milk instead of cow milk?	r(cookbook:milk, replacement, cookbook:goat_milk).	none	entity	☑	cookbook:goat_milk
				property name	☑	ingredientReplacement
				type name	☐	
				triple	☐	
3	How long it will take to make crepes?	r(cookbook:crepes, totalTime, Time).	none	entity	☐	
				property name	☐	
				type name	☐	
				triple	☑	<cookbook:crepes, totalTime, "30 min">

Fig. 1. Examples of annotated user's queries. This annotated queries are not part of the dataset.

The annotations were made using a unique table for each annotator as presented in Figure 1. Using the provided guidelines[7] the annotators had to write the associated Prolog query and to give the result of it, or if it was not possible, to give the elements missing in the KG that made the question unanswerable.

We decided to remove from the final dataset all the annotated user's queries where the annotator wrote that more than one element was missing in the KG. The first reason is that in this case, there can be multiple ways to represent the missing knowledge in the KG and to annotate the reasons why the query cannot be answered. In other words, the annotation would be subjective and the dataset would suffer from inconsistencies. Secondly, regarding machine learning algorithms, the tasks of identifying the elements missing will be much more complex if it has to be able to detected when multiple elements are missing for one user's query, as it corresponds to a multi-labelling task. However, we consider that the annotated user's queries that were removed from the dataset, will be

[7] The guidelines can be found here `https://perso.limsi.fr/rosset/resources/cooking_LL_QA.tar.gz`

useful anyway, either for detecting when a user's query cannot be answered, or in the future when a system will be mature enough.

Five Annotators, including PhD students and researchers from our institutions, participated in the annotation process. Each annotator had to know at least about the basis of Prolog. We expected to remove around 10% of the annotated data when multiple elements were missing, so we decided to annotate 110 user's queries to get at the end 100 annotated queries in the final dataset. To make it possible, each annotator had 22 user's queries to annotate. The annotation process was quite long, since the annotators had to check for each element if they exist in the KG and under which name. Depending on the knowledge on Prolog and on the cooking KG, the annotators needed from 5 minutes to 20 minutes to annotate one user's question. This time take into account the corrections needed. At the end one person was responsible of reviewing all the annotations and to correct them in order to have consistent data.

4.3 Description of the dataset

The original dataset is provided in the form of an Excel document. It contains all the annotations as presented in Figure 1 with the comments of annotators. The characteristics of the original dataset are presented in table 2. The final dataset is provided in the form of a json file[8]. The questions where more than one element was missing have been removed from this dataset. When a question contained typos, we replaced the question with the corrected one in the final dataset. The characteristics of the final dataset are presented in table 3.

#questions	110
answerable	40%
one element missing	38%
multiple element missing	22%

Table 2. Original dataset. Proportions of questions among some categories.

#questions	86
typo mistakes	09%
one element missing	49%
type of element missing	
entity	43%
type name	02%
property name	24%
triple	31%

Table 3. Final dataset. Proportions of questions among some categories.

5 Lessons learned through the creation process

We have observed that in the majority of cases, the questions that cannot be answered initially can become answerable after populating the knowledge graph with new entities, property names or triples. So the system can evolve over time. However, there is one situation where the system can't evolve easily: when it affects the structure of the data. For example, in the current version of the KG, the

[8] `https://perso.limsi.fr/rosset/resources/cooking_LL_QA.json`

information related to a recipe ingredient is just a triple, but several questions require it to be a tuple with additional information beyond the foodstuff (quantity or amount, possible replacement, textual description, etc.). This problem cannot be overcome by adding some triples, but altering the current structure of the recipe ingredients nodes.

After completing the annotation process we re-evaluate the questions that we annotated as not usable regarding our task. We came to the point that we actually filtered too many questions and determined that only 10% of the questions were not usable (against 27% previously). We also figured out that we underestimated the proportion of questions with more that one element missing (22% against 10% estimated). That is why the final dataset actually contains 86 annotated questions instead of 100.

6 Conclusion and future work

In a real exploitation environment, usual QA systems would provide an incorrect answer when the question refers to element that are missing in the KB. Thus we state that it is fundamental for lifelong learning QA systems to be able to handle jointly the two following problems: the translation of Natural Language (NL) questions into formal queries, and the detection and identification of missing knowledge that impact the way questions are answered. As no evaluation resources are yet available to address these problems, we presented in this paper a methodology for the creation of these resources. Moreover we publicly share the resulting resources, namely i) A cooking KG and (ii) the first version of a dataset containing a set of questions over the KG with the element missing in the KB if an answer cannot be found.

For future work we plan to collect and annotate more questions by taking advantage of lessons learned though the creation of the first version of the dataset.

Acknowledgments

This work has been supported by ERA-Net CHIST-ERA LIHLITH Project funded by ANR (France) project ANR-17-CHR2-0001-03, and AEI (Spain) projects PCIN-2017-085/AEI and RTI2018-096846-B-C21 (MCIU/AEI/FEDER, UE).

References

1. Bordes, A., Usunier, N., Chopra, S., Weston, J.: Large-scale Simple Question Answering with Memory Networks. CoRR (jun 2015)
2. Diefenbach, D., Lopez, V., Singh, K., Maret, P.: Core techniques of question answering systems over knowledge bases: a survey. Knowledge and Information Systems **55**(3), 529–569 (jun 2018)
3. He, X., Golub, D.: Character-Level Question Answering with Attention. In: In EMNLP 2016. pp. 1598–1607. Association for Computational Linguistics, Stroudsburg, PA, USA (2016)

4. Höffner, K., Walter, S., Marx, E., Usbeck, R., Lehmann, J., Ngonga Ngomo, A.C.: Survey on challenges of question answering in the semantic web. Semantic Web **8**(6), 895–920 (2017)
5. Joshi, M., Choi, E., Weld, D.S., Zettlemoyer, L.: TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. CoRR (may 2017)
6. Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. CoRR (sep 2019)
7. Lopez, V., Unger, C., Cimiano, P., Motta, E.: Evaluating question answering over linked data. Web Semantics Science Services And Agents On The World Wide Web **21**, 3–13 (2013)
8. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level. In: In WWW 2017. pp. 1211–1220. ACM Press, New York, New York, USA (2017)
9. Mazumder, S., Liu, B., Wang, S., Ma, N.: Lifelong and interactive learning of factual knowledge in dialogues. In: In SIGDIAL 2019. pp. 21–31. ACL, Stockholm, Sweden (Sep 2019)
10. Mohammed, S., Shi, P., Lin, J.: Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks. CoRR (dec 2017)
11. Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., Deng, L.: MS MARCO: A human generated MAchine reading COmprehension dataset. CEUR Workshop Proceedings **1773** (nov 2016)
12. Rajpurkar, P., Jia, R., Liang, P.: Know What You Don't Know: Unanswerable Questions for SQuAD. CoRR (jun 2018)
13. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., Le, Q.V.: XLNet: Generalized Autoregressive Pretraining for Language Understanding. CoRR (jun 2019)