A Knowledge Graph Question-Answering Platform Trained Independently of the Graph

Reham Omar, Ishika Dhall, Nadia Sheikh, and Essam Mansour

Concordia University, Canada
{fname.lname}@concordia.ca

Abstract. We will demonstrate KGQAn, a question-answering platform trained independently of KGs. KGQAn transforms a question into semantically equivalent SPARQL queries via a novel three-phase strategy based on natural language models trained generally for understanding and leveraging short English text. Without preprocessing or annotated questions on KGs, KGQAn outperformed the existing systems in KG question answering by an improvement of at least 33% in F1-measure and 61% in precision. During the demo, the audience will experience KGQAn for question answering on real KGs of topics of interest to them, such as DBpedia and OpenCitations Graph, and review the generated SPARQL queries and answers. A demo video is available online [1].

1 Introduction

There is growing adoption of knowledge graphs (KGs) across the industry on variant application domains, such as life science, media, e-commerce, and government, to integrate heterogeneous datasets. The RDF data model is widely utilized to store KGs due to RDF's simplicity, powerful query language (SPARQL), and extensions, such as RDF Schema and Web Ontology Language. These extensions help in formalizing constraints and semantics on top of RDF datasets. This knowledge formalization enables inferencing and reasoning to enrich KGs with new derived facts. Hence, there is an explosion in the number of RDF-based KGs, which have an on-line service (endpoint) receiving SPARQL queries via HTTP requests. Examples of these KGs are DBpedia [2] and Wikidata [3] (both are structured content of Wikipedia), Microsoft Academic Graph [4] (eight billion triples about scientific publications, authors, and institutions), OpenCitations [5], YAGO [6], and the UK Parliament [7]. These KGs are frequently updated.

The RDF data model represents ⟨`subject`, `predicate`, `object`⟩ triples, where a relationship (`predicate`) and an entity (`subject` or `object`) are identified by a URI. Forming structured SPARQL queries requires prior knowledge about the schema of the KG as well as the exact URIs of the entities and predicates. For example, to form the SPARQL query for Q: *find when did the Boston Tea Party take place and who was it led by*, the user needs to (i) be aware of the exact URIs

[1] https://rebrand.ly/kgqan_demo
[2] https://dbpedia.org/sparql
[3] https://query.wikidata.org/
[4] https://makg.org/
[5] https://opencitations.net/sparql
[6] https://yago-knowledge.org/sparql
[7] https://api.parliament.uk/sparql

of predicates, e.g., `<http://dbpedia.org/property/leadfigures>`, and entities, e.g., `<http://dbpedia.org/resource/Boston_Tea_Party>`, and (*ii*) identify basic graph patterns (BGP), e.g., a set of triple patterns connected through a common subject/object. Thus, forming a SPARQL query is challenging.

To easily explore KGs, KG question answering systems map a question into semantically equivalent SPARQL queries. Existing systems need thousands of fully annotated questions or require excessive preprocessing. DTQA [1] used about fifteen thousand fully annotated questions to train its models. Other systems access the entire KG in a preprocessing phase to build indexes, such as gAnswer [4], and encode semantics in KGs, such as WDAqua-core1 [2]. The complexity of the preprocessing phase is proportional to the KG size. The time complexity of the preprocessing phase in gAnswer is polynomial to the number of vertices in the KG [4]. A less data-intensive approach is taken by NLQSK [5], a primarily rule-based system. However, this system fails to outperform gAnswer on QALD-7 [10], which is less challenging than QALD-9 [9]. QALD-9 is a widely used question answering benchmark. Existing systems suffer from low accuracy and high false positives. For example, DTQA and gAnswer are the top-ranked systems for QALD-9 achieving F1-measure of 30.88 and 29.81 with precision 31.41 and 29.34, respectively, as reported at [9,1]. KGs are frequently updated, i.e., these systems will need to get more annotated questions or redo the preprocessing. Hence, there is a need for novel techniques that are trained independently of KGs.

We developed KGQAn, a question-answering platform trained independently of KGs, to address the above challenges. NLP-based models are effectively used to construct KGs from text.This fact inspired us to develop a three-phase strategy based on NLP models trained generally for understanding and leveraging a short English text. KGQAn outperforms the state-of-the-art systems by improving F1-measure and precision by least 33% and 61%, respectively. In this demo, Section 2 outlines the KGQAn architecture. Section 3 gives a glimpse on the evaluation. Section 4 explains the demo scenario and concludes.

2 The KGQAn System

We outline our three-phase strategy and demonstrate each phase as illustrated in Figure 1. Unlike existing systems, our strategy utilizes NLP models trained independently of the targeted KG, then uses lightweight SPARQL queries to annotate PGP's node and edges with corresponding vertices and predicates using semantic similarity models. KGQAn starts by extracting relation triples from the question to construct a phrase graph pattern (PGP). The relation triples are extracted from a question using our relation triple generator model pretrained for short English text. KGQAn fetches via SPARQL queries vertices and predicates, ranks them semantically to annotate the PGP with top-k ones. Finally, KGQAn executes the BGP queries with the highest rank, then filters the queries whose answers do not match the predicted answer data type.

The PGP Predictor extracts a set of relation triples patterns from the question to construct the PGP. A relation triple pattern is a triple of a relation phrase connecting two entities, e.g., a relation triple is ⟨`Boston Tea Party`, `take`

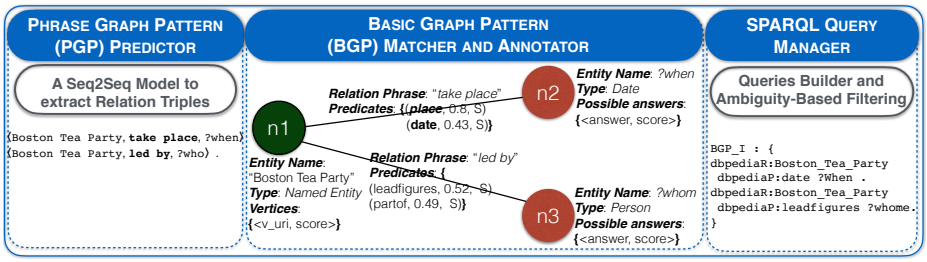

Fig. 1: The KGQAn main pipeline for the running example. The PGP extracted from Q has two relation triples with expected answer types date and person.

`place`, `?when`⟩ in Figure 1. Our triple generator model is a Sequence to Sequence (Seq2Seq) Deep Learning model based on BART [6] model. The BART model is a transformer model which can be used in different generation tasks such as machine translation and text summarization. We mapped the relation triples extraction task to a Seq2Seq generation task where the input is a short English text and the output are the triples extracted from this input. The models submitted in WEBNLG Challenge [8] attempted to solve this task. However, these models were trained on long English text which was not suitable for KGQAn. Due to the lack of datasets for the triples generation task for short English text, we prepared a manually annotated dataset, independent of KGs, where the source (input) is a short English text (Question) and the target (output) is the extracted triples. This dataset is built using 1000 questions collected from the LC-QuAD 1.0 benchmark [7].

Preparing our annotated dataset does not need the annotated SPARQL query from LC-QuAD. We only extract the English questions from LC-QuAD to annotate them. For example, for the input Q: *How many movies did Stanley Kubrick direct?*, the output is "Subject": Stanley Kubrick, "Object": ?unknown, "Predicate": direct. The dataset covers a wide variety of question types. In our relation triples generation model, we fine-tuned the BART large model to work on the triples generation task. First, the triples generation model is trained separately using the prepared dataset on a GPU machine. We trained the model for 3 epochs with a batch size of 4. Moreover, we used Adam optimizer with a learning rate of 0.0005, and a gelu activation function. The resulting trained model is saved to be integrated with KGQAn. After that KGQAn uses this trained model to extract the relation triples from the question under consideration. Then, the predicted triples are post-processed to construct the PGP. The generated triples are in (subject, predicate, object) format where the subject, object, and variables correspond to PGP nodes, and the relation phrases correspond to PGP edges as shown in Figure 1.

The BGP Matcher aims to annotate the PGP nodes and edges with candidate vertices and predicates in the target KG. The main idea is to prune the search space, i.e., the KG, by identifying a set of vertices that match syntactically the detected entities. The BGP Matcher then calculates the semantic affinity between the vertex's label and the entity to rank the top-k vertices. Our semantic affinity model is based on Word Embeddings [3]. For example, Table

[8] `https://webnlg-challenge.loria.fr/challenge_2020/`

Table 1: Top-k URIs of vertices matching "Boston Tea Party"

Vertex URI	Score
PREFIX dbpedia: `http://dbpedia.org/resource/`	
`dbpedia:Boston_Tea_Party`	1
`dbpedia:Boston_Tea_Party_(political_party)`	0.53
`dbpedia:Tea_Party_movement`	0.5

1 shows the top three vertices used to annotate the node "Boston Tea Party". VURI denotes the list of vertices annotating a particular entity. For the lack of space, we do not show the label retrieved with each vertex.

To annotate the PGP edges, KGQAn fetches, for each vertex v, a set of predicates connected to v when v is a subject and when it is an object. The PGP is an undirected graph, as the direction of the edge is based on the actual triple in the KG. KGQAn annotates each edge in the PGP with a set of tuples. Each tuple indicates a specific predicate, its direction, and its semantic affinity score to the extracted predicate. We formulate intermediate SPARQL queries searching for (i) a subject whose label matches a set of keywords, or the subject is of a particular type to fetch v, and (ii) a set of predicates connected with a given v as subject or object.

SPARQL Query Manager: This component generates the top-k SPARQL queries by traversing the annotated PGP to select the top-k basic graph patterns (BGP) semantically equivalent to the question. Our algorithm generates all the combinations of ⟨`subject`, `predicate`, `object`⟩ using these lists. Our `Ambiguity-Based Ranking and Filtering` module adjusts the final score of a BGP based on the semantic affinity between the predicate and the predicted answer type. Furthermore, the result of the queries is filtered depending on the answer data type. KGQAn filters the query whose result does not match the predicted answer data type.

3 Experimental Evaluation

The state-of-the-art systems, such as DTQA, gAnswer, and WDAqua, were evaluated using the most recent Challenge on Question Answering over Linked Data (QALD-9) [9,1]. gAnswer and WDAqua are ranked first and second in the QALD-9 challenge, respectively [9]. DTQA was also evaluated using a subset of LC-QuAD 1.0 [8] test questions. Both LC-QuAD and QALD-9 use DBpedia. Recently, DTQA slightly outperformed gAnswer in QALD-9. We used the common benchmark, i.e., QALD-9, in our evaluation. DTQA and gAnswer trained their models based on DBpedia's vertices and predicates, indirectly (by seeing thousands of questions and their corresponding SPARQL queries) or directly (by accessing the full graph in the preprocessing phase).

For the word embedding, we used a GloVe model pre-trained on 16B tokens from a large English corpus. We deployed the DBpedia dataset used with QALD-9 at a virtuoso SPARQL endpoint running at a remote VM. KGQAn submits SPARQL queries via HTTP calls to the remote SPARQL endpoint. We evaluated KGQAn using the QALD-9 test dataset, i.e., the 150 questions on DBpedia. Unlike other systems, KGQAn did not use the the QALD-9 training

Table 2: Evaluating QALD 9 Testing Questions (150 Questions)

Systems	Precision	Recall	Macro F1
WDAqua	26.09	26.7	24.99
gAnswer	29.34	32.68	29.81
DTQA	31.41	32.16	30.88
KGQAn	**50.61**	34.67	**41.15**

dataset to tune its performance. Without preprocessing on DBpedia or annotated questions, KGQAn significantly outperformed DTQA and gAnswer, especially in terms of precision and F1-measure, as shown in Table 2. Moreover, KGQAn is efficient in terms of time. The average time is less than 3 seconds for the end-to-end KGQAn pipeline, including the execution time of all the intermediate SPARQL queries, which fetch the candidate URIs for vertices and predicates, plus the execution of the final top-k semantically equivalent queries.

4 Demonstration and Conclusion

In this demo, we will use real datasets in the order of billions RDF triples from DBpedia and OpenCitations Graph. KGQAn uses models trained independently of the KG. Thus, we will also allow participants to explore a KG of their choice if the KG has a public SPARQL endpoint. We integrated our Seq2Seq model into a pipeline of lightweight queries to efficiently map a question to its semantically equivalent SPARQL queries. The demo will give a chance to discuss exciting research ideas inspired by such a pipeline to solve similar problems.

References

1. Abdelaziz, I., Ravishankar, S., Kapanipathi, P., et al.: A semantic parsing and reasoning-based approach to knowledge base question answering (2021)
2. Diefenbach, D., Singh, K.D., Maret, P.: Wdaqua-core1: A question answering service for RDF knowledge bases. In: Companion of the The Web Conference (2018)
3. Goikoetxea, J., Agirre, E., Soroa, A.: Single or multiple? combining word representations independently learned from text and wordnet. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
4. Hu, S., Zou, L., Yu, J.X., Wang, H., Zhao, D.: Answering natural language questions by subgraph matching over knowledge graphs. TKDE **30**(5), 824–837 (2018)
5. Hu, X., Duan, J., Dang, et al.: Natural language question answering over knowledge graph: the marriage of sparql query and keyword search. Knowledge and Information Systems (2021)
6. Lewis, M., Liu, Y., Goyal, N., et al.: Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461 (2019)
7. Trivedi, P., Maheshwari, et al.: Lc-quad: A corpus for complex question answering over knowledge graphs. In: International Semantic Web Conference (2017)
8. Trivedi, P., Maheshwari, G., Dubey, M., Lehmann, J.: Lc-quad: A corpus for complex question answering over knowledge graphs. In: ISWC. pp. 210–218 (2017)
9. Usbeck, R., Gusmita, R.H., Ngomo, A.N., Saleem, M.: 9th challenge on question answering over linked data (QALD-9). CEUR Workshop, vol. 2241 (2018)
10. Usbeck, R., Ngomo, A.C.N., Haarmann, et al.: 7th open challenge on question answering over linked data (qald-7). In: Semantic web evaluation challenge (2017)