UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Message Passing for Complex Question Answering over Knowledge Graphs

Vakulenko, S.; Fernandez Garcia, J.D.; Polleres, A.; de Rijke, M.; Cochez, Michael

DOI
10.48550/arXiv.1908.06917
10.1145/3357384.3358026

Publication date
2019

Document Version
Author accepted manuscript

Published in
CIKM'19

Link to publication

Citation for published version (APA):
Vakulenko, S., Fernandez Garcia, J. D., Polleres, A., de Rijke, M., & Cochez, M. (2019). Message Passing for Complex Question Answering over Knowledge Graphs. In *CIKM'19: Proceedings of the 28th ACM International Conference on Information & Knowledge Management : November 3-7, 2019, Beijing, China* (pp. 1431-1440). Association for Computing Machinery. https://doi.org/10.48550/arXiv.1908.06917, https://doi.org/10.1145/3357384.3358026

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:07 Oct 2026

Message Passing for Complex Question Answering over Knowledge Graphs

Svitlana Vakulenko
Vienna University of Economics and Business
Vienna, Austria
svitlana.vakulenko@wu.ac.at

Javier David Fernandez Garcia
Vienna University of Economics and Business
Vienna, Austria
jfernand@wu.ac.at

Axel Polleres
Vienna University of Economics and Business
Vienna, Austria
axel.polleres@wu.ac.at

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
derijke@uva.nl

Michael Cochez
Fraunhofer FIT
Sankt Augustin, Germany
michaelcochez@gmail.com

arXiv:1908.06917v1 [cs.CL] 19 Aug 2019

ABSTRACT

Question answering over knowledge graphs (KGQA) has evolved from simple single-fact questions to complex questions that require graph traversal and aggregation. We propose a novel approach for complex KGQA that uses unsupervised message passing, which propagates confidence scores obtained by parsing an input question and matching terms in the knowledge graph to a set of possible answers. First, we identify entity, relationship, and class names mentioned in a natural language question, and map these to their counterparts in the graph. Then, the confidence scores of these mappings propagate through the graph structure to locate the answer entities. Finally, these are aggregated depending on the identified question type.

This approach can be efficiently implemented as a series of sparse matrix multiplications mimicking joins over small local subgraphs. Our evaluation results show that the proposed approach outperforms the state-of-the-art on the LC-QuAD benchmark. Moreover, we show that the performance of the approach depends only on the quality of the question interpretation results, i.e., given a correct relevance score distribution, our approach always produces a correct answer ranking. Our error analysis reveals correct answers missing from the benchmark dataset and inconsistencies in the DBpedia knowledge graph. Finally, we provide a comprehensive evaluation of the proposed approach accompanied with an ablation study and an error analysis, which showcase the pitfalls for each of the question answering components in more detail.

CCS CONCEPTS

• **Mathematics of computing** → **Probabilistic reasoning algorithms**; • **Information systems** → **Uncertainty**; **Question answering**; • **Computing methodologies** → **Semantic networks**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '19, November 03–07, 2019, Beijing, China
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

KEYWORDS

Question answering, Knowledge graph, Message passing, Spreading activation, Associative retrieval, Approximate reasoning

ACM Reference Format:
Svitlana Vakulenko, Javier David Fernandez Garcia, Axel Polleres, Maarten de Rijke, and Michael Cochez. 2019. Message Passing for Complex Question Answering over Knowledge Graphs. In *Proceedings of CIKM '19: The 28th ACM International Conference on Information and Knowledge Management (CIKM '19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The amount of data shared on the Web grows every day [14]. Information retrieval systems are very efficient but they are limited in terms of the representation power for the underlying data structure that relies on an index for a single database table, i.e., a homogeneous collection of textual documents that share the same set of attributes, e.g., web pages or news articles [30]. Knowledge graphs (KGs), i.e., graph-structured knowledge bases, such as DBpedia [27] or Wikidata [48], can interlink datasets with completely different schemas [4]. Moreover, SPARQL is a very expressive query language that allows us to retrieve data from a KG that matches specified graph patterns [19]. Query formulation in SPARQL is not easy in practice since it requires knowledge of which datasets to access, their vocabulary and structure [12]. Natural language interfaces can mitigate these issues, making data access more intuitive and also available for the majority of lay users [20, 22]. One of the core functionalities for this kind of interfaces is question answering (QA), which goes beyond keyword or boolean queries, but also does not require knowledge of a specialised query language [46].

QA systems have been evolving since the early 1960s with early efforts in the database community to support natural language queries by translating them into structured queries [see, e.g., 6, 17, 51]. Whereas a lot of recent work has considered answering questions using unstructured text corpora [38] or images [16], we consider the task of answering questions using information stored in KGs. KGs are an important information source as an intermediate representation to integrate information from different sources and different modalities, such as images and text [7]. The resulting models are at the same time abstract, compact, and interpretable [50].

Question answering over knowledge graphs (KGQA) requires matching an input question to a subgraph, in the simplest case matching a single labeled edge (*triple*) in the KG, a task also called *simple* question answering [5]. The task of *complex* question answering is defined in contrast to simple KGQA and requires matching more than one triple in the KG [45]. Previously proposed approaches to complex KGQA formulate it as a subgraph matching task [1, 29, 44], which is an NP-hard problem (by reduction to the subgraph isomorphism problem) [53], or attempt to translate a natural language question into template-based SPARQL queries to retrieve the answer from the KG [8], which requires a large number of candidate templates [42].

We propose an approach to complex KGQA, called QAmp, based on an unsupervised message-passing algorithm, which allows for efficient reasoning under uncertainty using text similarity and the graph structure. The results of our experimental evaluation demonstrate that QAmp is able to manage uncertainties in interpreting natural language questions, overcoming inconsistencies in a KG and incompleteness in the training data, conditions that restrict applications of alternative supervised approaches.

A core aspect of QAmp is in disentangling reasoning from the question interpretation process. We show that uncertainty in reasoning stems from the question interpretation phase alone, meaning that under correct question interpretations QAmp will always rank the correct answers at the top. QAmp is designed to accommodate uncertainty inherent in perception and interpretation processes via confidence scores that reflect natural language ambiguity, which depends on the ability to interpret terms correctly. These ranked confidence values are then aggregated through our message-passing in a well-defined manner, which allows us to simultaneously consider multiple alternative interpretations of the seed terms, favoring the most likely interpretation in terms of the question context and relations modeled within the KG. Rather than iterating over all possible orderings, we show how to evaluate multiple alternative question interpretations in parallel via efficient matrix operations.

Another assumption of QAmp that proves useful in practice is to deliberately disregard subject-object order, i.e., edge directions in a knowledge graph, thereby treating the graph as undirected. Due to relation sparsity, this model relaxation turns out to be sufficient for most of the questions in the benchmark dataset. We also demonstrate that due to insufficient relation coverage of the benchmark dataset any assumption on the correct order of the triples in the KG is prone to overfitting. More than one question-answer example per relation is required to learn and evaluate a supervised model that predicts relation directionality.

Our evaluation on LC-QuAD[1] [45], a recent large-scale benchmark for complex KGQA, shows that QAmp significantly outperforms the state-of-the-art, without the need to translate a natural language question into a formal query language such as SPARQL. We also show that QAmp is interpretable in terms of activation paths, and simple, effective and efficient at the same time. Moreover, our error analysis demonstrates limitations of the LC-QuAD benchmark, which was constructed using local graph patterns.

The rest of the paper is organized as follows. Section 2 summarizes the state of the art in KGQA. Section 3 presents our approach, QAmp, with particular attention to the *question interpretation* and *answer inference* phases. In Section 4, we evaluate QAmp on the LC-QuAD dataset, providing a detailed ablation, scalability and error study. Finally, Section 6 concludes and lists future work.

2 RELATED WORK

The most commonly used KGQA benchmark is the SimpleQuestions [5] dataset, which contains questions that require identifying a single triple to retrieve the correct answers. Recent results [37] show that most of these simple questions can be solved using a standard neural network architectures. This architecture consists of two components: (1) a conditional random fields (CRF) tagger with GloVe word embeddings for subject recognition given the text of the question, and (2) a bidirectional LSTM with FastText word embeddings for relation classification given the text of the question and the subject from the previous component. Approaches to simple KGQA cannot easily be adapted to solving complex questions, since they rely heavily on the assumption that each question refers to only one entity and one relation in the KG, which is no longer the case for complex questions. Moreover, complex KGQA also requires matching more complex graph patterns beyond a single triple.

Since developing KGQA systems requires solving several tasks, namely entity, relation and class linking, and afterwards query building, they are often implemented as independent components and arranged into a single pipeline [9]. Frameworks such as QALL-ME [11], OKBQA [24] and Frankenstein [41], allow one to share and reuse those components as a collaborative effort. For example, Frankenstein includes 29 components that can be combined and interchanged [43]. However, the distribution of the number of components designed for each task is very unbalanced. Most of the components in Frankenstein support entity and relation linking, 18 and 5 components respectively, while only two components perform query building [42].

There is a lack of diversity in approaches that are being considered for retrieving answers from a KG. OKBQA and Frankenstein both propose to translate natural language questions to SPARQL queries and then use existing query processing mechanism to retrieve answers.[2] We show that using matrix algebra approaches is more efficient in case of natural language processing than traditional SPARQL-based approaches since they are optimized for parallel computation, thereby allowing us to explore multiple alternative question interpretations at the same time [21, 23].

Query building approaches involve query generation and ranking steps [29, 52]. These approaches essentially consider KGQA as a subgraph matching task [1, 29, 44], which is an NP-hard problem (by reduction to the subgraph isomorphism problem) [53]. In practice, Singh et al. [42] report that the question building components of Frankenstein fail to process 46% questions from a subset of LC-QuAD due to the large number of triple patterns. The reason is that most approaches to query generation are template-based [8] and complex questions require a large number of candidate templates [42]. For example, WDAqua [8] generates 395 SPARQL queries as possible interpretations for the question "Give me philosophers born in Saint Etienne."

[1] http://lc-quad.sda.tech

[2] http://doc.okbqa.org/query-generation-module/v1/

In summary, we identify the query building component as the main bottleneck for the development of KGQA systems and propose QAmp as an alternative to the query building approach. Also, the pipeline paradigm is inefficient since it requires KG access first for disambiguation and then again for query building. QAmp accesses the KG only to aggregate the confidence scores via graph traversal after question parsing and shallow linking that matches an input question to labels of nodes and edges in the KG.

The work most similar to ours is the spreading activation model of Treo [13], which is also a no-SPARQL approach based on graph traversal that propagates relatedness scores for ranking nodes with a cut-off threshold. Treo relies on POS tags, the Stanford dependency parser, Wikipedia links and TF/IDF vectors for computing semantic relatedness scores between a question and terms in the KG. Despite good performance on the QALD 2011 dataset, the main limitation of Treo is an average query execution time of 203s [13]. In this paper we show how to scale this kind of approach to large KGs and complement it with machine learning approaches for question parsing and word embeddings for semantic expansion.

Our approach overcomes the limitations of the previously proposed graph-based approach in terms of efficiency and scalability, which we demonstrate on a compelling benchmark. We evaluate QAmp on LC-QuAD [45], which is the largest dataset used for benchmarking complex KGQA. WDAqua is our baseline approach, which is the state-of-the-art in KGQA as the winner of the most recent Scalable Question Answering Challenge (SQA2018) [33]. Our evaluation results demonstrate improvements in precision and recall, while reducing average execution time over the SPARQL-based WDAqua, which is also orders of magnitude faster than results reported for the previous graph-based approach Treo.

There is other work on KGQA that uses embedding queries into a vector space [18, 49]. The benefit of our graph-based approach is in preserving the original structure of the KG that can be used for executing precise formal queries and answering ambiguous natural language questions. The graph structure also makes the results traceable and, therefore, interpretable in terms of relevant paths and subgraphs in comparison with vector space operations.

QAmp uses message passing, a family of approaches that were initially developed in the context of probabilistic graphical models [25, 35]. Graph neural networks trained to learn patterns of message passing have recently shown to be effective on a variety of tasks [2, 15], including KG completion [39]. We show that our unsupervised message passing approach performs well on complex question answering and helps to overcome sampling biases in the training data, which supervised approaches are prone to.

3 APPROACH

QAmp, our KGQA approach, consists of two phases: (1) question interpretation, and (2) answer inference. In the question interpretation phase we identify the sets of entities and predicates that we consider relevant for answering the input question along with the corresponding confidence scores. In the second phase these confidence scores are propagated and aggregated directly over the structure of the KG, to provide a confidence distribution over the set of possible answers. Our notion of KG is inspired by common concepts from the Resource Description Framework (RDF) [40], a standard representation used in many large-scale knowledge graphs, e.g., DBpedia and Wikidata:

Definition 3.1. We define a (knowledge) *graph* $K = \langle E, G, P \rangle$ as a tuple that contains sets of *entities* E (nodes) and *properties* P, both represented by Unique Resource Identifiers (URIs), and a set of directed labeled edges $\langle e_i, p, e_j \rangle \in G$, where $e_i, e_j \in E$ and $p \in P$.

The set of edges G in a KG can be viewed as a (blank-node-free) RDF graph, with subject-predicate-object triples $\langle e_i, p, e_j \rangle$. In analogy with RDFS, we refer to a subset of entities $C \subseteq E$ appearing as objects of the special property `rdf:type` as *Classes*. We also refer to classes, entities and properties collectively as *terms*. We ignore RDF literals, except for `rdfs:labels` that are used for matching questions to terms in KG.

The task of *question answering over a knowledge graph* (KGQA) is: given a natural language question Q and a knowledge graph K, produce the correct answer A, which is either a subset of entities in the KG $A \subseteq E$ or a result of a computation performed on this subset, such as the number of entities in this subset (`COUNT`) or an assertion (`ASK`). These types of questions are the most frequent in existing KGQA benchmarks [5, 45, 47]. In the first phase QAmp maps a natural language question Q to a structured model q, which the answer inference algorithm will operate on then.

3.1 Question interpretation

To produce a question model q we follow two steps: (1) *parse*, which extracts references (entity, predicate and class mentions) from the natural language question and identifies the question type; and (2) *match*, which assigns each of the extracted references to a ranked list of candidate entities, predicates and classes in the KG.

Effectively, a complex question requires answering several sub-questions, which may depend on or support each other. A dependence relation between the sub-questions means that an answer A^1 to one of the questions is required to produce the answer A^2 for the other question: $A^2 = f(A^1, K)$. We call such complex questions *compound* questions and match the sequence in which these questions should be answered to *hops* (in the context of this paper, one-variable graph patterns) in the KG. Consider the sample compound question in Fig. 1, which consists of two hops: (1) find the car types assembled in Broadmeadows Victoria, which have a hardtop style, (2) find the company, which produces these car types. There is an intermediate answer (the car types with the specified properties), which is required to arrive at the final answer (the company).

Accordingly, we define (compound) questions as follows:

Definition 3.2. A *question model* is a tuple $q = \langle t_q, Seq_q \rangle$, where $t_q \in T$ is a question type required to answer the question Q, and $Seq_q = (\langle E^i, P^i, C^i \rangle)_{i=1}^{h}$ is a sequence of h hops over the KG, E^i is a set of entity references, P^i – a set of property references, C^i – a set of class references relevant for the i-hop in the graph, and T – a set of question types, such as {`SELECT`, `ASK`, `COUNT`}.

Hence, the question in Fig. 1 can be modeled as: $\langle$`SELECT`, $(\langle E^1 =$ {"hardtop", "Broadmeadows, Victoria"}, $P^1 =$ {"assembles", "style"}, $C^1 =$ {"cars"}$\rangle$, $\langle E^2 = \emptyset, P^2 =$ {"company"}, $C^2 = \emptyset \rangle) \rangle$, where E^i, P^i, C^i refer to the entities, predicates and classes in hop i.

Further, we describe how the question model q is produced by parsing the input question Q, after which we match references in q to entities and predicates in the graph K.

Parsing. Given a natural language question Q, the goal is to classify its type t_q and parse it into a sequence Seq_q of reference sets according to Definition 3.2. Question type detection is implemented as a supervised classification model trained on a dataset of annotated question-type pairs that learns to assign an input question to one of the predefined types $t_q \in T$.

We model reference (mention) extraction Seq_q as a sequence labeling task [26], in which a question is represented as a sequence of tokens (words or characters). Then, a supervised machine learning model is trained on an annotated dataset to assign labels to tokens, which we use to extract references to entities, predicates and classes. Moreover, we define the set of labels to group entities, properties and classes referenced in the question into h hops.

Matching. Next, the question model (Definition 3.2) is updated with an *interpreted question model* $I(q) = (t_q, SEQ_q)$ in which each component of Seq_q is represented by sets of pairs from $(E \cup P \cup C) \times [0, 1]$ obtained by matching the references to concrete terms in K (by their URIs) as follows: for each entity (or property, class, resp.) reference in Seq_q, we retrieve a ranked list of most similar entities from the KG along with the matching confidence score.

Fig. 1 also shows the result of this matching step on our example. For instance, the property references for the first hop are replaced by the set of candidate URIs: $P^1 = \{P_1^1, P_2^1\} \in SEQ_q$ within $I(q)$, where $P_1^1 = \{(\texttt{dbo:assembly}, 0.9), (\texttt{dbp:assembly}, 0.9)\}$, $P_2^1 = \{(\texttt{dbo:bodyStyle}, 0.5)\}$.

3.2 Answer inference

Our answer inference approach iteratively traverses and aggregates confidence scores across the graph based on the initial assignment from $I(q)$. An answer set A^i, i.e., a set of entities along with their confidence scores $E \times [0, 1]$, is produced after each hop i and used as part of the input to the next hop $i+1$, along with the terms matched for this hop in $I(q)$, i.e., $SEQ_q(i+1) = \langle E^{i+1}, P^{i+1}, C^{i+1} \rangle$. The entity set A^h produced after the last hop h can be further transformed to produce the final answer: $A_q = f_{t_q}(A^h)$ via an aggregation function $f_{t_q} \in F$ from a predefined set of available aggregation functions F defined for each of the question types $t_q \in T$. We compute the answer set A^i for each hop inductively in two steps: (1) *subgraph extraction* and (2) *message passing*.

Subgraph extraction. This step refers to the retrieval of relevant triples from the KG that form a subgraph. Thus, the URIs of the matched entities and predicates in the query are used as seeds to retrieve the triples in the KG that contain at least one entity (in subject or object position), and one predicate from the corresponding reference sets. Therefore, the extracted subgraph will contain n entities, which include all entities from E^i and the entities adjacent to them through properties from P^i.

The subgraph is represented as a set of k adjacency matrices with n entities in the subgraph: $\mathbb{S}^{k \times n \times n}$, where k is the total number of matched property URIs. There is a separate $n \times n$ matrix for each of the k properties used as seeds, where $\mathbb{S}_{pij} = 1$ if there is an edge labeled p between the entities i and j , and 0 otherwise. All adjacency matrices are symmetric, because $I(q)$ does not model edge directionality, i.e., it treats K as undirected. Diagonal entries are assigned 0 to ignore self loops.

Algorithm 1 Message passing for KGQA

Input: adjacency matrices of the subgraph $\mathbb{S}^{k \times n \times n}$, entity $\mathbb{E}^{l \times n}$ and property reference activations $\mathbb{P}^{m \times k}$
Output: answer activations vector $A \in \mathbb{R}^n$

1: $W^n, N_P^n, \mathbb{Y}_{\mathbb{E}}{}^{l \times n} = \emptyset$
2: **for** $P_j \in \mathbb{P}^{m \times k}, j \in \{1, ..., m\}$ **do**
3: $\mathbb{S}_j = \bigoplus_{i=1}^{k} P_j \otimes \mathbb{S}$ ▷ property update
4: $\mathbb{Y} = \mathbb{E} \oplus \otimes \mathbb{S}_j$ ▷ entity update
5: $W = W + \bigoplus_{i=1}^{l} \mathbb{Y}_{ij}$ ▷ sum of all activations
6: $N_{P_j} = \sum_{i=1}^{l} 1$ if $\mathbb{Y}_{ij} > 0$ else 0
7: $\mathbb{Y}_{\mathbb{E}} = \mathbb{Y}_{\mathbb{E}} \oplus \mathbb{Y}$ ▷ activation sums per entity
8: **end for**
9: $W = 2 \cdot W/(l + m)$ ▷ activation fraction
10: $N_E = \sum_{i=1}^{l} 1$ if $\mathbb{Y}_{\mathbb{E}ij} > 0$ else 0
11: **return** $A = (W \oplus N_E \oplus N_P)/(l + m + 1)$

Message passing. The second step of the answer inference phase involves message passing,[3] i.e., propagation of the confidence scores from the entities E^i and predicates P^i, matched in the question interpretation phase, to adjacent entities in the extracted subgraph. This process is performed in three steps, (1) *property update*, (2) *entity update*, and (3) *score aggregation*. Algorithm 1 summarizes this process, detailed as follows.

For each of m property references $P_j \in \mathbb{P}^{m \times k}, j \in \{1, \ldots, m\}$ where $m = |P^i|$, we

(1) *select* the subset of adjacency matrices from $\mathbb{S}^{k \times n \times n}$ for the property URIs if $p_{ij} > 0$, where $p_{ij} \in P_j$, and propagate the confidence scores to the edges of the corresponding adjacency matrices via element-wise multiplication. Then, all adjacency matrices are combined into a single adjacency matrix $\mathbb{S}_j^{n \times n}$, which contains all of their edges with the sum of confidence scores if edges overlap (*property update*: line 3, Algorithm 1).
(2) *perform* the main message-passing step via the sum-product update, in which the confidence scores from l entity references, where $l = |E^i|$, are passed over to the adjacent entities via all edges in $\mathbb{S}_j^{n \times n}$ (*entity update*: line 4, Algorithm 1).
(3) *aggregate* the confidence scores for all n entities in the subgraph into a single vector A by combining the sum of all confidence scores with the number of entity and predicate reference sets, which received non-zero confidence score. The intuition behind this *score aggregation* formula (line 11, Algorithm 1) is that the answers that received confidence from the majority of entity and predicate references in the question should be preferred. The computation of the answer scores for our running example is illustrated in Fig. 2.

The minimal confidence for the candidate answer is regulated by a threshold to exclude partial and low-confidence matches. Finally, we also have an option to filter answers by considering only those entities in the answer set A^i that have one of the classes in C^i.

[3] The pseudocode of the message passing algorithm is presented in Algorithm 1.

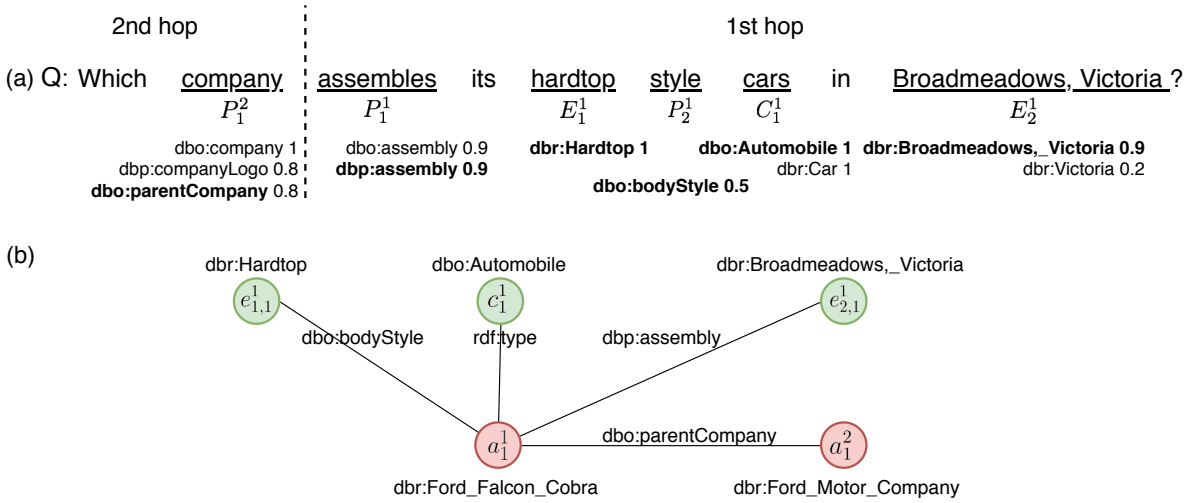

Figure 1: (a) A sample question Q highlighting different components of the question interpretation model: references and matched URIs with the corresponding confidence scores, along with (b) the illustration of a sample KG subgraph relevant to this question. The URIs in bold are the correct matches corresponding to the KG subgraph.

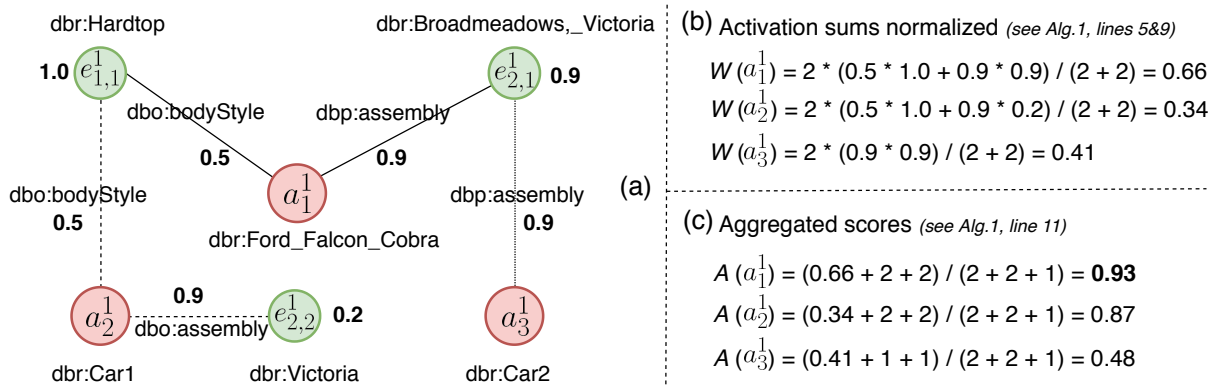

Figure 2: (a) A sample subgraph with three entities as candidate answers, (b) their scores after predicate and entity propagation, and (c) the final aggregated score.

The same procedure is repeated for each hop in the sequence using the corresponding URI activations for entities, properties and classes modeled in $SEQ_q(i) = \langle E^i, P^i, C^i \rangle$ and augmented with the intermediate answers produced for the previous hop A^{i-1}. Lastly, the answer to the question A_q is produced based on the entity set A^h, which is either returned 'as is' or put through an aggregation function f_{t_q} conditioned on the question type t_q.

4 EVALUATION SETUP

We evaluate QAmp, our KGQA approach, on the LC-QuAD dataset of complex questions constructed from the DBpedia KG [45]. First, we report the evaluation results of the end-to-end approach, which incorporates our message-passing algorithm in addition to the initial question interpretation (question parser and matching functions). Second, we analyze the fraction and sources of errors produced by different KGQA components, which provides a comprehensive perspective on the limitations of the current state-of-the-art for KGQA, the complexity of the task, and limitations of the benchmark. Our implementation and evaluation scripts are open-sourced.[4]

Baseline. We use WDAqua [8] as our baseline; to the best of our knowledge, the results produced by WDAqua are the only published results on the end-to-end question answering task for the LC-QuAD benchmark to date. It is a rule-based framework that integrates several KGs in different languages and relies on a handful of SPARQL query patterns to generate SPARQL queries and rank them as likely question interpretations. We rely on the evaluation results reported by the authors [8]. WDAqua results were produced for the full LC-QuAD dataset, while other datasets were used for tuning the approach.

Metrics. We follow the standard evaluation metrics for the end-to-end KGQA task, i.e., we report precision (P) and recall (R) macro-averaged over all questions in the dataset, and then use them to compute the F-measure (F). Following the evaluation setup of the QALD-9 challenge [47] we assign both precision and recall equal to

[4] https://github.com/svakulenk0/KBQA

Table 1: Dataset statistics: number of questions across the train and test splits; number of complex questions that reference more than one triple; number of complex questions that require two hops in the graph through an intermediate answer-entity.

	Questions		
Split	All	Complex	Compound
all	4,998 (100%)	3,911 (78%)	1,982 (40%)
train	3,999 (80%)	3,131 (78%)	1,599 (40%)
test	999 (20%)	780 (78%)	383 (38%)

0 for every question in the following cases: (1) for SELECT questions, no answer (empty answer set) is returned, while there is an answer (non-empty answer set) in the ground truth annotations; (2) for COUNT or ASK questions, an answer differs from the ground truth; (3) for all questions, the predicted answer type differs from the ground truth. In the ablation study, we also analyze the fraction of questions with errors for each of the components separately, where an error is a not exact match with the ground-truth answer.

Hardware. We used a standard commodity server to train and evaluate QAmp: Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, RAM 16 GB DDR4 SDRAM 2400 MHz, 240 GB SSD, NVIDIA GP102 GeForce GTX 1080 Ti.

4.1 The LC-QuAD dataset

The LC-QuAD dataset[5] [45] contains 5K question-query pairs that have correct answers in the DBpedia KG (2016-04 version). The questions were generated using a set of SPARQL templates by seeding them with DBpedia entities and relations, and then paraphrased by human annotators. All queries are of the form ASK, SELECT, and COUNT, fit to subgraphs with diameter of at most 2-hops, contain 1–3 entities and 1–3 properties.

We used the train and test splits provided with the dataset (Table 1). Two queries with no answers in the graph were excluded. All questions are also annotated with ground-truth reference spans[6] to evaluate performance of entity linking and relation detection [9].

4.2 Implementation details

Our implementation uses the English subset of the official DBpedia 2016-04 dump losslessly compressed into a single HDT file[7] [10]. HDT is a state-of-the-art compressed RDF self-index, which scales linearly with the size of the graph and is, therefore, applicable to very large graphs in practice. This KG contains 1B triples, more than 26M entities (*dbpedia.org* namespace only) and 68,687 predicates. Access to the KG for subgraph extraction and class constraint look-ups is implemented via the Python HDT API.[8] This API builds an additional index [31] to speed up all look-ups, and consumes the HDT mapped in disk, with a ~3% memory footprint.[9]

[5] https://github.com/AskNowQA/LC-QuAD
[6] https://github.com/AskNowQA/EARL
[7] http://fragments.dbpedia.org/hdt/dbpedia2016-04en.hdt
[8] https://github.com/Callidon/pyHDT
[9] Overall, DBpedia 2016-04 takes 18GB in disk, and 0.5GB in main memory.

Our end-to-end KGQA solution integrates several components that can be trained and evaluated independently. The pipeline includes two supervised neural networks for (1) question type detection and (2) reference extraction; and unsupervised functions for (3) entity and (4) predicate matching, and (5) message passing.

Parsing. Question type detection is implemented as a bi-LSTM neural-network classifier trained on pairs of question and type. We use another biLSTM+CRF neural network for extracting references to entities, classes and predicates for at most two hops using the set of six labels: {"E1", "P1", "C1", "E2", "P2", "C2"}. Both classifiers use GloVe word embeddings pre-trained on the Common Crawl corpus with 840B tokens and 300 dimensions [36].

Matching. The labels of all entities and predicates in the KG (`rdfs:label` links) are indexed into two separate catalogs and embedded into two separate vector spaces using the English FastText model trained on Wikipedia [3]. We use two ranking functions for matching and assigning the corresponding confidence scores: index-based for entities and embedding-based for predicates. The index-based ranking function uses BM25 [30] to calculate confidence scores for the top-500 matches on the combination of n-grams and Snowball stems.[10] Embedding-based confidence scores are computed using the Magnitude library[11] [34] for the top-50 nearest neighbors in the FastText embedding space.

Many entity references in the LC-QuAD questions can be handled using simple string similarity matching techniques; e.g., 'companies' can be mapped to "http://dbpedia.org/ontology/Company". We built an ElasticSearch (Lucene) index to efficiently retrieve such entity candidates via string similarity to their labels. The entity labels were automatically generated from entity URIs by stripping the domain part of the URI and lower-casing, e.g., entity "http://dbpedia.org/ontology/Company" received the label "company" to better match question words. LC-QuAD questions also contain more complex paraphrases of the entity URIs that require semantic similarity computation beyond fuzzy string matching, such as "movie" refers to "http://dbpedia.org/ontology/Film", "stockholder" to "http://dbpedia.org/property/owner" or "has kids" to 'http://dbpedia.org/ontology/child'. We embeded entity and predicate labels with FastText [3] to detect semantic similarities beyond string matching.

Index-based retrieval scales much better than nearest neighbour computation in the embedding space, which is a crucial requirement for the 26M entity catalog. In our experiments, syntactic similarity was sufficient for entity matching in most of the cases, while property matching required capturing more semantic variations and greatly benefited from using pre-trained embeddings.

5 EVALUATION RESULTS

Table 2 shows the performance of QAmp on the KGQA task in comparison with the results previously reported by Diefenbach et al. [8]. There is a noticeable improvement in recall (we were able to retrieve answers to 50% of the benchmark questions), while maintaining a comparable precision score. For the most recent QALD challenge the guidelines were updated to penalize systems that miss the correct answers, i.e., that are low in recall, which gives

[10] https://www.elastic.co/guide/en/elasticsearch/reference/current
[11] https://github.com/plasticityai/magnitude

Table 2: Evaluation results. (*) P of the WDAqua baseline is estimated from the reported precision of 0.59 for answered questions only. Runtime is reported in seconds per question as an average across all questions in the dataset. The distribution of runtimes for QAmp is Min: 0.01, Median: 0.67 Mean: 0.72, Max: 13.83

Approach	P	R	F	Runtime
WDAqua	0.22*	0.38	0.28	1.50 s/q
QAmp (our approach)	0.25	0.50	0.33	0.72 s/q

a clear signal of its importance for this task [47]. While it is often trivial for users to filter out a small number of incorrect answers that stem from interpretation ambiguity, it is much harder for users to recover missing correct answers. Indeed, we showed that QAmp is able to identify correct answers that were missing even from the benchmark dataset since they were overlooked by the benchmark authors due to sampling bias.

5.1 Ablation study

Table 3 summarizes the results of our ablation study for different setups. We report the fraction of all questions that have at least one answer that deviates from the ground truth (*Total* column), questions with missing term matches (*No match*) and other errors. Revised errors is the subset of other errors that were considered as true errors in the manual error analysis.

Firstly, we make sure that the relaxations in our question interpretation model hold true for the majority of questions in the benchmark dataset (95%) by feeding all ground truth entity, class and property URIs to the answer inference module (Setup 1 in Table 3). We found that only 53 test questions (5%) require one to model the exact order of entities in the triple, i.e., subject and predicate positions. These questions explicitly refer to a hierarchy of entity relations, such as `dbp:doctoralStudents` and `dbp:doctoralAdvisor` (see Figure 3[12, 13]), and their directionality has to be interpreted to correctly answer such questions. We also recovered a set of correct answers missing from the benchmark for relations that are symmetric by nature, but were considered only in one direction by the benchmark, e.g., `dbo:related`, `dbo:associatedBand`, and `dbo:sisterStation` (see Figure 4).

These results indicate that a more complex question model attempting to reflect structural semantics of a question in terms of the expected edges and their directions (parse graph or lambda calculus) is likely to fall short when trained on this dataset: 53 sample questions are insufficient to train a reliable supervised model that can recognize relation directions from text, which explains poor results of a slot-matching model for subgraph ranking reported on this dataset [28].

There were only 8 errors (1%) due to the wrong question type detected caused by misspelled or grammatically incorrect questions (row 2 in Table 3). Next, we experimented with removing class constraints and found that although they generally help to

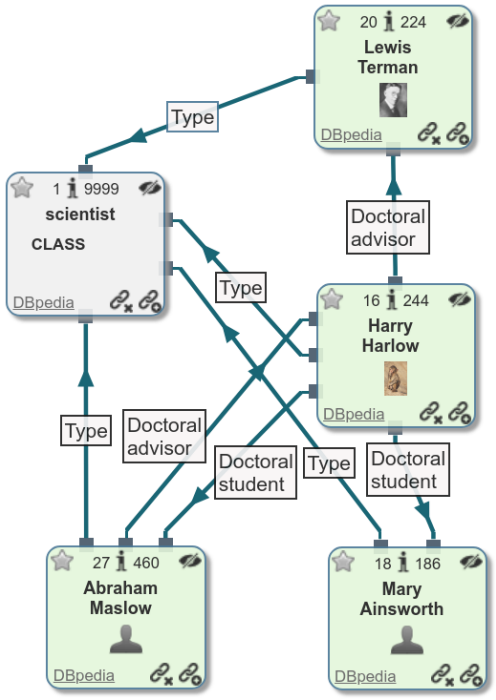

Figure 3: *Directed relation* example (`dbp:doctoralStudents` and `dbp:doctoralAdvisor` hierarchy) that requires modeling directionality of the relation. LC-QuAD question #3267: "Name the scientist whose supervisor also supervised Mary Ainsworth?" (correct answer: Abraham Maslow) can be easily confused with a question: "Name the scientist who supervised also the supervisor of Mary Ainsworth?" (correct answer: Lewis Terman). LC-QuAD benchmark is not suitable for evaluating directionality interpretations, since only 35 questions (3.5%) of the LC-QuAD test split use relations of this type, which explains high performance results of QAmp that treats all relation as undirected.

filter out incorrect answers (row 3) our matching function missed many correct classes even using the ground-truth spans from the benchmark annotations (row 4).

The last four evaluation setups (5–8) in Table 3 show the errors from parsing and matching reference spans to entities and predicates in the KG. Most errors were due to missing term matches (10–34% of questions), which indicates that the parsing and matching functions constitute the bottleneck in our end-to-end KGQA. Even with the ground-truth span annotations for predicate references the performance is below 0.6 (34% of questions), which indicates that relation detection is much harder than the entity linking task, which is in line with results reported by Dubey et al. [9] and Singh et al. [42].

The experiments marked **GT span+** were performed by matching terms to the KG using the ground-truth span annotations, then down-scaling the confidence scores for all matches and setting the confidence score of the match used in the ground-truth query to the maximum confidence score of 1. In this setup, all correct answers according to the benchmark were ranked at the top, which

[12] The numbers on top of each entity show its number of predicates and triples.
[13] All sample graph visualizations illustrating different error types discovered in the LC-QuAD dataset in Figure 3, 4, 6 were generated using the LODmilla web tool, http://lodmilla.sztaki.hu [32], with data from the DBpedia KG.

Table 3: Ablation study results. (*) Question model results set the optimal performance for our approach assuming that the question interpretation is perfectly aligned with the ground-truth annotations. We then estimate additional (new) errors produced by each of the KGQA components separately. The experiments marked with GT use the term URIs and question types extracted from the ground truth queries. GT span$^+$ uses spans from the ground-truth annotations and then corrects the distribution of the matched entities/properties to mimic correct question interpretation with a low-confidence tail with alternative matches. PR (Parsed Results) stands for predictions made by question parsing and matching models (see Section 4.2).

	Setup	Question interpretation				P	R	F	Questions with errors		
		Q. type	Entity	Property	Class				Total	New errors	
										No match	Other→Revised
1	Question model*	GT				0.97	0.99	0.98	9%	–	9% → 5%
2	Question type	PR	GT			0.96	0.98	0.97	10%	–	1% → 1%
3	Ignore classes	GT			None	0.94	0.99	0.96	14%	–	5% → 3%
4	Classes GT span$^+$	GT			GT span$^+$	0.89	0.92	0.90	17%	8%	–
5	Entities GT span$^+$	GT	GT span$^+$	GT		0.85	0.88	0.86	20%	10%	1% → 1%
6	Entities PR	GT	PR	GT		0.64	0.74	0.69	46%	27%	10% → 5%
7	Predicates GT span$^+$	GT		GT span$^+$	GT	0.56	0.59	0.57	48%	34%	5% → 3%
8	Predicates PR	GT		PR	GT	0.36	0.53	0.43	74%	34%	31% → 19%

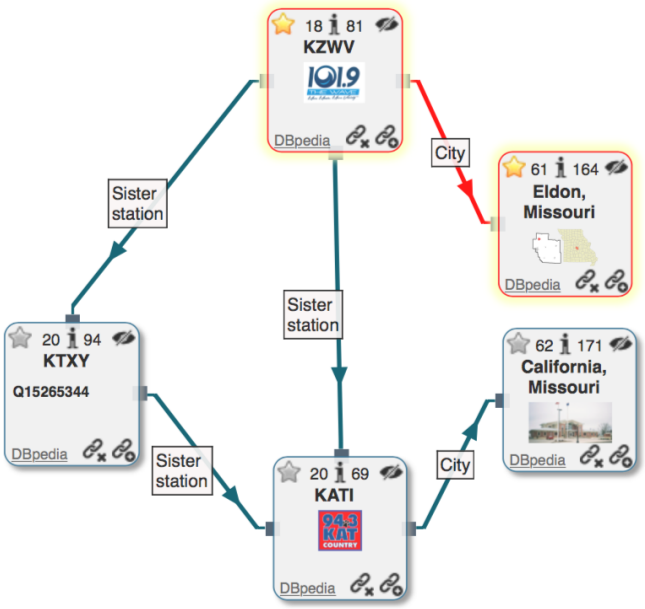

Figure 4: *Undirected relation* example (`dbo:sisterStation`) that reflects bi-directional association between the adjacent entities (Missouri radio stations). LC-QuAD question #4486: "In which city is the sister station of KTXY located?" (correct answer: `dbr:California,Missouri`, `dbr:Missouri`; missing answer: `dbr:Eldon,Missouri`). DBpedia does not model bi-directional relations and the relation direction is selected at random in these cases. LC-QuAD does not reflect bi-directionality either by picking only one of the directions as the correct one and rejecting correct solutions (`dbr:KZWY` → `dbr:Eldon,Missouri`). QAmp was able to retrieve this false negative sample due to the default undirectionality assumption built into the question interpretation model.

demonstrates the correctness of the message passing and score aggregation algorithm.

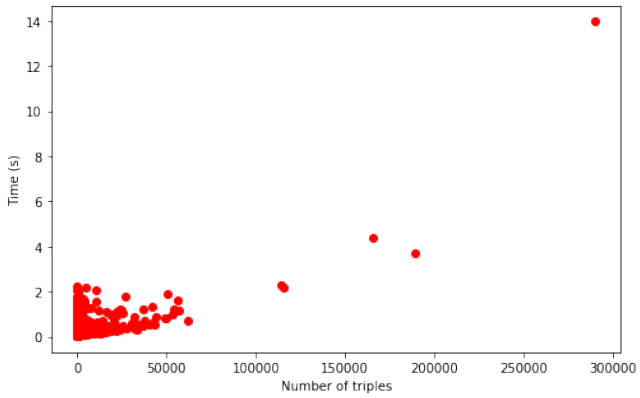

Figure 5: Processing times per question from the LC-QuAD test split (Min: 0.01s Median: 0.68s Mean: 0.72s Max: 13.97s)

5.2 Scalability analysis

As we reported in Table 2, QAmp is twice as fast as the WDAqua baseline using a comparable hardware configuration. Figure 5 shows the distribution of processing times and the number of examined triples per question from the LC-QuAD test split. The results are in line with the expected fast retrieval of HDT [10], which scales linearly with the size of the graph. Most of the questions are processed within 2 seconds (with a median and mean around 0.7s), even those examining more than 50K triples. Note that only 10 questions took more than 2 seconds to process and 3 of them took more than 3 seconds. These outliers end up examining a large number of alternative interpretations (up to 300K triples), which could be prevented by setting a tighter threshold. Finally, it is worth mentioning that some questions end up with no results (i.e., 0 triples accessed), but they can take up to 2 seconds for parsing and matching.

5.3 Error analysis

We sampled questions with errors (P < 1 or R < 1) for each of the setups and performed an error analysis for a total of 206 questions. Half of the errors were due to the incompleteness of the benchmark dataset and inconsistencies in the KG (column *Revised* in Table 3).

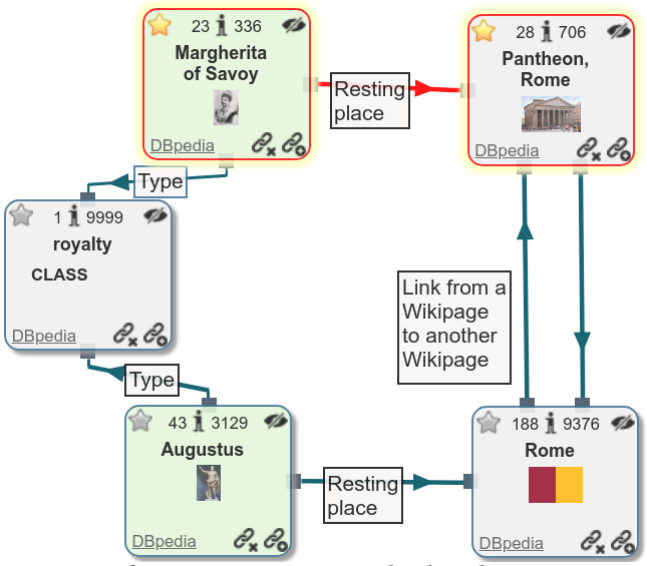

Figure 6: *Alternative entity* example that demonstrates a missing answer when only a single correct entity URI is considered (`dbr:Rome` and not `dbr:Pantheon,Rome`). LC-QuAD question #261: "Give me a count of royalties buried in Rome?" (correct answer: `dbr:Augustus`; missing answer: `dbr:Margherita_of_Savoy`). QAmp was able to retrieve this false negative sample due to the string matching function and retaining a list of alternative URIs per entity mention.

Since the benchmark provides only a single SPARQL query per question that contains a single URI for each entity, predicate and class, all alternative though correct matches are missing, e.g., the gold-truth query using `dbp:writer` will miss `dbo:writer`, or match all `dbo:languages` triples but not `dbo:language`, etc.

QAmp was able to recover many such cases to produce additional correct answers using: (1) missing or alternative class URIs, e.g., `dbr:Fire_Phone` was missing from the answers for technological products manufactured by Foxconn since it was annotated as a `device`, and not as an `information appliance`; (2) related or alternative entity URIs, e.g., the set of royalties buried in `dbo:Rome` should also include those buried in `dbr:PantheonRome` (see Figure 6); (3) alternative properties, e.g., `dbo:hometown` as `dbo:birthPlace`.

We discovered alternative answers due to the *majority vote* effect, when many entities with low confidence help boost a single answer. Majority voting can produce a best-effort guess based on the data in the KG even if the correct terms are missing from the KG or could not be recovered by the matching function, e.g., *"In which time zone is Pong Pha?"* – even if *Pong Pha* is not in the KG many other locations with similar names are likely to be located in the same geographic area.

Overall, our evaluation results indicate that the answer set of the LC-QuAD benchmark can be used only as a seed to estimate recall but does not provide us with a reliable metric for precision. Attempts to further improve performance on such a dataset can lead to learning the biases embedded in the construction of the dataset, e.g., the set of relations and their directions. QAmp is able to mitigate this pitfall by resorting to unsupervised message passing that collects answers from all local subgraphs containing terms matching the input question, in parallel.

6 CONCLUSION

We have proposed QAmp, a novel approach for complex KGQA using message passing, which sets the new state-of-the-art results on the LC-QuAD benchmark for complex question answering. We have shown that QAmp is scalable and can be successfully applied to very large KGs, such as DBpedia, which is one of the biggest cross-domain KGs. QAmp does not require supervision in the answer inference phase, which helps to avoid overfitting and to discover correct answers missing from the benchmark due to the limitations of its construction. Moreover, the answer inference process can be explained by the extracted subgraph and the confidence score distribution. QAmp requires only a handful of hyper-parameters to model confidence thresholds in order to stepwise filter partial results and trade off recall for precision.

QAmp is built on a basic assumption of considering edges as undirected in the graph, which proved reasonable and effective in our experiments. The error analysis revealed that, in fact, symmetric edges were often missing in the KG, i.e., the decision on the order of entities in KG triples is made arbitrarily and is not duplicated in the reverse order. However, there is also a (small) subset of relations, e.g., hierarchy relations, for which relation direction is essential.

Question answering over KGs is hard due to (1) ambiguities stemming from question interpretation, (2) inconsistencies in knowledge graphs, and (3) challenges in constructing a reliable benchmark, which motivate the development of robust methods able to cope with uncertainties and also provide assistance to end-users in interpreting the provenance and confidence of the answers.

QAmp is not without limitations. It is designed to handle questions where the answer is a subset of entities or an aggregate based on this subset, e.g., questions for which the expected answer is a subset of properties in the graph, are currently out of scope. An important next step is to use QAmp to improve the recall of the benchmark dataset by complementing the answer set with missing answers derived from relaxing the dataset assumptions. Recognizing relation directionality is an important direction for future work, which requires extending existing benchmark datasets and the addition of more cases where an explicit order is required to retrieve correct answers. Another direction is to improve predicate matching, which is the weakest component of the proposed approach as identified in our ablation study. Finally, unsupervised message passing can be adopted for other tasks that require uncertain reasoning on KGs, such as knowledge base completion, text entailment, summarization, and dialogue response generation.

Acknowledgments

This work was supported by the EU H2020 programme under the MSCA-RISE agreement 645751 (RISE_BPM), the Austrian Research Promotion Agency (FFG) under projects CommuniData (855407) and Cityspin (861213), Ahold Delhaize, the Association of Universities in the Netherlands (VSNU), and the Innovation Center for Artificial Intelligence (ICAI). All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Jun-Wei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-Based Question Answering with Knowledge Graph. In *COLING 2016*. 2503–2514.

[2] Peter W. Battaglia, Jessica B. Hamrick, et al. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[4] Piero Andrea Bonatti, Stefan Decker, Axel Polleres, and Valentina Presutti. 2018. Knowledge Graphs: New Directions for Knowledge Representation on the Semantic Web (Dagstuhl Seminar 18371). *Dagstuhl Reports* 8, 9 (2018), 29–111.

[5] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale Simple Question Answering with Memory Networks. *CoRR* abs/1506.02075 (2015). arXiv:1506.02075

[6] Wim Bronnenberg, Harry Bunt, Jan Landsbergen, Remko Scha, Wijnand Schoenmakers, and Eric van Utteren. 1980. The question answering system Phliqa1. In *Natural Language Question Answering Systems*, L. Bolc (Ed.). MacMillan, 217–305.

[7] Fabrício F. de Faria, Ricardo Usbeck, Alessio Sarullo, Tingting Mu, and André Freitas. 2018. Question Answering Mediated by Visual Clues and Knowledge Graphs. In *WWW*. 1937–1939.

[8] Dennis Diefenbach, Andreas Both, Kamal Deep Singh, and Pierre Maret. 2018. Towards a Question Answering System over the Semantic Web. *CoRR* abs/1803.00832 (2018). arXiv:1803.00832

[9] Mohnish Dubey, Debayan Banerjee, Debanjan Chaudhuri, and Jens Lehmann. 2018. EARL: Joint Entity and Relation Linking for Question Answering over Knowledge Graphs. In *ISWC 2018*. 108–126.

[10] Javier D. Fernández, Miguel A. Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias. 2013. Binary RDF Representation for Publication and Exchange (HDT). *J. Web Semant.* 19 (2013), 22–41.

[11] Óscar Ferrández, Christian Spurk, Milen Kouylekov, Iustin Dornescu, Sergio Ferrández, Matteo Negri, Rubén Izquierdo, David Tomás, Constantin Orasan, Guenter Neumann, Bernardo Magnini, and José Luis Vicedo González. 2011. The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture. *J. Web Semant.* 9, 2 (2011), 137–145.

[12] André Freitas, Edward Curry, João Gabriel Oliveira, and Seán O'Riain. 2012. Querying Heterogeneous Datasets on the Linked Data Web: Challenges, Approaches, and Trends. *IEEE Internet Computing* 16, 1 (2012), 24–33.

[13] André Freitas, João Gabriel Oliveira, Seán O'Riain, João Carlos Pereira da Silva, and Edward Curry. 2013. Querying linked data graphs using semantic relatedness: A vocabulary independent approach. *Data Knowl. Eng.* 88 (2013), 126–141.

[14] Amir Gandomi and Murtaza Haider. 2015. Beyond the hype: Big data concepts, methods, and analytics. *International journal of information management* 35, 2 (2015), 137–144.

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, et al. 2017. Neural Message Passing for Quantum Chemistry. In *ICML 2017*. 1263–1272.

[16] Yash Goyal, Tejas Khot, et al. 2017. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *CVPR 2017*.

[17] Bert F. Green, Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. 1963. Baseball: An automatic question answerer. In *Computers and Thought*. McGraw-Hill, 219–224.

[18] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding Logical Queries on Knowledge Graphs. In *NeurIPS*. 2030–2041.

[19] Steve Harris and Andy Seaborne. 2013. SPARQL 1.1 Query Language. W3C Recommendation. (March 2013).

[20] Gary G Hendrix. 1982. Natural-language interface. *Computational Linguistics* 8, 2 (1982), 56–61.

[21] Fuad Jamour, Ibrahim Abdelaziz, and Panos Kalnis. 2018. A demonstration of MAGiQ: matrix algebra approach for solving RDF graph queries. *Proc. the VLDB Endowment* 11, 12 (2018), 1978–1981.

[22] Esther Kaufmann and Abraham Bernstein. 2007. How Useful Are Natural Language Interfaces to the Semantic Web for Casual End-Users?. In *ISWC*. 281–294.

[23] Jeremy Kepner, Peter Aaltonen, David A. Bader, Aydin Buluç, Franz Franchetti, John R. Gilbert, Dylan Hutchison, Manoj Kumar, Andrew Lumsdaine, Henning Meyerhenke, Scott McMillan, Carl Yang, John D. Owens, Marcin Zalewski, Timothy G. Mattson, and José E. Moreira. 2016. Mathematical foundations of the GraphBLAS. In *HPEC*. 1–9.

[24] Jin-Dong Kim, Christina Unger, Axel-Cyrille Ngonga Ngomo, André Freitas, YoungGyun Hahm, Jiseong Kim, Gyu-Hyun Choi, Jeonguk Kim, Ricardo Usbeck, Myoung-Gu Kang, and Key-Sun Choi. 2017. OKBQA: an Open Collaboration Framework for Development of Natural Language Question-Answering over Knowledge Bases. In *ISWC*.

[25] Daphne Koller, Nir Friedman, and Francis Bach. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT press.

[26] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *ICML 2001*. 282–289.

[27] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, et al. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[28] Gaurav Maheshwari, Priyansh Trivedi, et al. 2018. Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs. *arXiv preprint arXiv:1811.01118* (2018).

[29] Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. 2018. Learning to Rank Query Graphs for Complex Question Answering over Knowledge Graphs. *arXiv preprint arXiv:1811.01118* (2018).

[30] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. 2010. Introduction to information retrieval. *Natural Language Engineering* 16, 1 (2010), 100–103.

[31] Miguel A Martínez-Prieto, Mario Arias Gallego, and Javier D Fernández. 2012. Exchange and consumption of huge RDF data. In *ESWC*. 437–452.

[32] András Micsik, Sándor Turbucz, and Attila Györök. 2014. LODmilla: a Linked Data Browser for All. In *Posters&Demos SEMANTiCS 2014*, Sack Harald, Filipowska Agata, Lehmann Jens, and Hellmann Sebastian (Eds.). CEUR-WS.org, 31–34.

[33] Giulio Napolitano, Ricardo Usbeck, and Axel-Cyrille Ngonga Ngomo. 2018. The Scalable Question Answering Over Linked Data (SQA) Challenge 2018. In *SemWebEval Challenge at ESWC*. 69–75.

[34] Ajay Patel, Alexander Sands, Chris Callison-Burch, and Marianna Apidianaki. 2018. Magnitude: A Fast, Efficient Universal Vector Embedding Utility Package. In *EMNLP 2018*. 120–126.

[35] Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc.

[36] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *EMNLP 2014*. 1532–1543.

[37] Michael Petrochuk and Luke Zettlemoyer. 2018. SimpleQuestions Nearly Solved: A New Upperbound and Baseline Approach. In *EMNLP 2018*. 554–558.

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *EMNLP 2016*. 2383–2392.

[39] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. In *ESWC*. 593–607.

[40] Guus Schreiber and Yves Raimond. 2014. RDF 1.1 Primer. W3C Note. (June 2014).

[41] Kuldeep Singh, Andreas Both, Arun Sethupat Radhakrishna, and Saeedeh Shekarpour. 2018. Frankenstein: A Platform Enabling Reuse of Question Answering Components. In *ESWC*. 624–638.

[42] Kuldeep Singh, Ioanna Lytra, Arun Sethupat Radhakrishna, Saeedeh Shekarpour, Maria-Esther Vidal, and Jens Lehmann. 2018. No One is Perfect: Analysing the Performance of Question Answering Components over the DBpedia Knowledge Graph. *CoRR* abs/1809.10044 (2018).

[43] Kuldeep Singh, Arun Sethupat Radhakrishna, Andreas Both, Saeedeh Shekarpour, Ioanna Lytra, Ricardo Usbeck, Akhilesh Vyas, Akmal Khikmatullaev, Dharmen Punjani, Christoph Lange, Maria-Esther Vidal, Jens Lehmann, and Sören Auer. 2018. Why Reinvent the Wheel: Let's Build Question Answering Systems Together. In *WWW*. 1247–1256.

[44] Daniil Sorokin and Iryna Gurevych. 2018. Modeling Semantics with Gated Graph Neural Networks for Knowledge Base Question Answering. In *COLING 2018*. 3306–3317.

[45] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. 2017. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In *ISWC*. 210–218.

[46] Christina Unger, André Freitas, and Philipp Cimiano. 2014. An Introduction to Question Answering over Linked Data. In *Reasoning Web*. 100–140.

[47] Ricardo Usbeck, Ria Hari Gusmita, Axel-Cyrille Ngonga Ngomo, and Muhammad Saleem. 2018. 9th Challenge on Question Answering over Linked Data. In *QALD at ISWC*. 58–64.

[48] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[49] Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. 2018. Towards Empty Answers in SPARQL: Approximating Querying with RDF Embedding. In *ISWC*. 513–529.

[50] Xander Wilcke, Peter Bloem, and Victor De Boer. 2017. The knowledge graph as the default data model for learning on heterogeneous knowledge. *Data Science* (2017), 1–19.

[51] William Aaron Woods. 1977. Lunar rocks in natural English: Explorations in natural language question answering. In *Linguistic Structures Processing*, A. Zampoli (Ed.). Elsevier North-Holland, 521–569.

[52] Hamid Zafar, Giulio Napolitano, and Jens Lehmann. 2018. Formal Query Generation for Question Answering over Knowledge Bases. In *ESWC*. 714–728.

[53] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD 2014*. 313–324.