Leveraging Knowledge Graph for Open-domain Question Answering

Jose Ortiz Costa
Department of Computer Science
San Francisco State University
San Francisco, USA
jortizco@mail.sfsu.edu

Anagha Kulkarni
Department of Computer Science
San Francisco State University
San Francisco, USA
ak@sfsu.edu

***Abstract*—Rich and comprehensive knowledge graphs (KG) of the Web, such as, Google KG, NELL, and Diffbot KG, are becoming increasingly prevalent and powerful as the underlying AI technology is rapidly progressing. In this work, we leverage this ongoing advancement for the task of answering questions posed from any domain and any type (factoid and non-factoid). We present a framework for knowledge graph based question answering systems, KGQA, and experiment with an instance of this framework that employs Diffbot KG. The unique features offered by KGs, such as, rapid query response time, connections between related graph objects, and structured information, are used to design a QA system that is effective and efficient.**

***Index Terms*—Automated Question Answering, Diffbot Knowledge Graph, Information Retrieval, Natural Language Processing**

I. Introduction

Automatic question answering problem (QA) has a long and rich history [3], [25], [28]. Early QA systems focused on narrow domains, and used hand-built rules and knowledge bases to *understand* the question, and then to answer it. With the advent of WWW, and developments in Natural Language Processing (NLP), Information Retrieval (IR), Data Mining (DM), and Machine Learning (ML) the QA systems have become increasingly less constrained. Open-domain QA systems that leverage WWW for the breadth of data it offers, are being extensively researched [2], [23], [24], [26], [27], [31]. The high-volume and the data redundancy offered by WWW is leveraged by many of these QA systems to compensate for the lack of structure to the data. However, there is renewed interest in using structured data, specifically knowledge graphs, for the QA problem [1], [6], [8], [11], [21], [22], [32], [33]. Unlike the previous KGs, however, the newer KGs are built from the data on the WWW using Data Mining (DM) and Machine Learning (ML) techniques. Examples of such KGs are Google KG [19], NELL [12], and Diffbot KG [18]. Advances in Deep Learning and Computer Vision are rapidly improving the depth, breadth, and connectivity of the KGs being created from the WWW. As such, KGs are becoming powerful tools for tasks, such as, answering questions from any domain.

Driven by these observations we propose a framework for knowledge graph based question answering approaches. To develop this QA framework we harness the unique properties of KGs: 1. the data redundancy that propagates from WWW to the KG, 2. easy access to the links between data objects, 3. the structure overlaid on the data, 4. ability to run queries against the KG extremely efficiently, and 5. dynamic and up-to-date nature of the KG. The first component of our system exploits data redundancy and low query latency to design a light-weight and versatile approach for transforming the user question into multiple queries. To obtain the answer-bearing data object in the second module, we leverage the structure and the connectivity in the data. In addition to other signals, we employ the recency property of the KGs to select the final answer. In this work, we experiment with a specific instance of this framework where the Diffbot KG is used to power the QA approach (KGQA).

II. KGQA: Knowledge Graph Based Question Answering Approach

As shown in Figure 1, our system is structured as a pipeline of four main components: Query Preprocessing Module (QPM), Multi-query Formulation Module (MQFM), KG Object Extraction Module (KGOEM), and Answer Extraction and Selection Module (AESM). The first module, QPM, performs basic textual and grammatical processing steps on the original user question. MQFM, generates multiple queries from the cleansed version of the question. These queries are

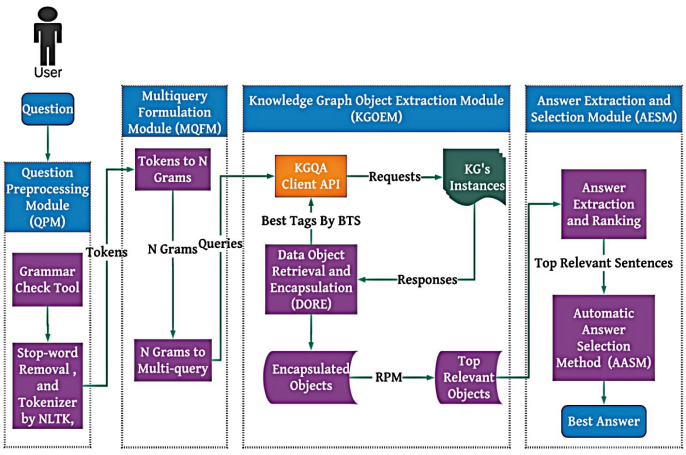

Fig. 1. KGQA System Architecture Diagram

then executed against the KG by the third module, KGOEM, and the extracted information is passed on to the last module. AESM then extracts short and focused candidate answers, from which the final answer is selected. Each of these components are described in details next. Note that only one of these modules, KGOEM, interfaces with the knowledge graph, and all the other modules are agnostic to the specific KG being used by the system. Thus the proposed approach can be viewed as a framework for KG based QA systems where other knowledge graphs can be employed by adapting just one module of the framework. We will use the following question as a running example throughout this section to illustrate the function of each module: *"How talls is Mount McKinley?"*.

A. Question Preprocessing Module (QPM)

This module performs light-weight preprocessing of the question that consists of four steps: filtering out sentences without question mark, running basic grammar check and correction, tokenization, and stop-words removal. An open-source Python library is employed for grammar check and correction [14] of the question. If there are any corrections suggested by the library, then they are always accepted and used to update the question. The original question from the running example is altered to *"How tall is Mount McKinley?"*, by this step. Next, tokenization splits the question into an **ordered** set of words using the tokenizer tool of Natural Language Library Toolkit (NLTK) [20], where the order among the tokens is dictated by their order of appearance in the original question. The high-frequency and low-information bearing words, such as, *the, and*, are removed from the set in the stop-words removal step. The default stop-words list provided by NLTK is employed for this filtering step. The resulting ordered set of tokens is represented as: $T = (t_1, ..., t_m)$. For the running example the ordered set is $T = (\text{“}tall\text{”}, \text{“}Mount\text{”}, \text{“}McKinley\text{”})$, and $m = 3$.

B. Multi-Query Formulation Module (MQFM)

This module is organized into two subcomponents: (1) The first step generates n-grams from the the ordered set of tokens provided by the previous module. (2) The second step creates multiple queries from those n-grams.

From tokens to n-grams: The ordered set of tokens $T = (t_1, ...t_m)$, is used to create set $G = \{g_1, ..., g_i, ..., g_m\}$ where each element of the set G, g_i, is a set itself that represents grams (i.e. unigrams, bigrams, trigrams, etc) of length i. Each set g_i is constructed by enumerating all sequences of consecutive i tokens from T. The size of each g_i set is specified by: $|g_i| = m - i + 1$, and thus the size of G is $|G| = \sum_{i=1}^{n} |g_i|$. For the running example, the generated set is: $G = \{g_1, g_2, g_3\}$, where:

- $g_1 = \{$*("tall"), ("Mount"), ("McKinley")*$\}$
- $g_2 = \{$*("tall Mount"), ("Mount McKinley")*$\}$
- $g_3 = \{$*("tall Mount McKinley")*$\}$.

From n-grams to queries: A set of queries are constructed in this step: $Q = \{q_1, ..., q_m\}$ as follows. The first query, q_1, consists of a boolean conjunctive query composed on all the unigrams in set g_1. The remaining queries, q_i, are built using two sets of grams: g_i, and g_1. For each gram $g_i^j \in g_i$ (say, "Mount McKinley"), a sub-query, q_i^j is defined as a boolean conjunction of the gram, g_i^j, and all the unigrams from g_1 that are not in g_i^j (e.g. "tall"). Field-restriction operators are then added to each query such that g_i^j is matched against the document title field, and the unigrams are matched against the document text field. The motivation is to identify documents where the central topic as captured by the title field matches the g_i^j gram. A boolean disjunction on all such sub-queries, q_i^j, creates the query, q_i. Using this approach, the set of three queries generated for the running example are as follows: $Q = \{q_1, q_2, q_3\}$, where:

- $q_1 =$ *("tall" AND "Mount" AND "McKinley")*
- $q_2 =$ *((title:"tall Mount" AND text:"McKinley") OR (title:"Mount McKinley" AND text:"tall"))*
- $q_3 =$ *("tall Mount McKinley")*.

C. Knowledge Graph Object Extraction Module (KGOEM)

The main goal of the KG Object Extraction Module is to retrieve the most relevant data objects from the Knowledge Graph using the queries from the previous section. In this work we employ a large, rich, and highly-responsive knowledge graph powered by Diffbot [18], that are organized into millions of types using machine learning, and computer vision techniques. We leverage this KG to obtain answer-bearing content from the Web. This is accomplished in two steps: 1. Data object retrieval and ranking, and 2. Data object selection.

*1) **Data Object Retrieval and Ranking**:* This step is responsible for selecting a few promising data objects for every query generated by the previous module. The linked nature of the KG, and the retrieved data objects are leveraged jointly to identify & retrieve additional data objects that were not retrieved by the query. Diffbot provides an array of standard APIs to obtain data from the KG. We use Diffbot's *search API* to run the set of queries generated in MQFM (Q) against the KG. For each query, Diffbot retrieves multiple data objects in JSON format. For each data object Diffbot provides a confidence score that is an estimate of the relevance of the object to the query. Summary statistics, such as, number of hits, number of documents in the collection are also provided. The returned objects can be of different types (article, discussion, video, image, product), and a different set of fields are returned for each type. We have developed Custom APIs to process and extract textual fields from each object type that are of value to the task. These are *caption* fields for *video* type, *title* for image type, *discussions* for *post* type, and *description* fields for product type. The *article* object type is processed more deeply (described next) since it is the most text-heavy type.

In addition to the article title and content, Diffbot provides a list of *tags* for each article [17]. The tags are entities that have been extracted from the article content and co-referenced with other data sources, such as, DBpedia. For our running example, one of the tags associated with one of the retrieved article objects is *Denali*, the new name of *Mount McKinley*. As such, these tags can provide valuable additional information

that can reduce the *vocabulary gap* between the question and the relevant content. Diffbot also provides the number of hyperlinks associated with each retrieved object (web-page). The tags and links model the connectivity of the object, and thus can be viewed as proxies for the quality and authority of the object. Based on this observation we formulate the following object scoring function:

$$S(o) = a * cf(o) + b * nt(o) + c * atc(o) + d * nl(o)$$

where $cf(o)$ is the confidence score of object o. $nt(o)$ is the number of *matching* tags associated with o. A tag is considered as matching when at least one of the query terms appears in the tag label. Diffbot provides an occurrence frequency of each tag label in the object's content. The average frequency over matching tags for o is provided by $atc(o)$. $nl(o)$ is the number of hyperlinks associated with o.

The scores assigned by this function are employed to rank all the retrieved objects for a query. Furthermore, the distribution of these scores is leveraged to infer how many of the top ranked objects should be selected for further analysis. Specifically, let $O = (o_1, o_2, ..., o_n)$ be the ordered set of retrieved objects for a query, and let $x = S(o_1) - S(o_2)$, which is the difference in scores of the top two ranked objects. For $i = 3, .., n$, if $S(o_{i-1}) - S(o_i) > 2 * x$, then discard objects $o_i, o_{(i+1)}, ..., o_n$. This approach retains a few highly scored objects for each query.

Tag Harvesting: For each entity tag, Diffbot provides two additional pieces of information: 1. an URI to a reference article, and 2. a confidence score based on tag's relevance to the article content. We harvest the URI field using Diffbot's *article API*. The article API first analyzes the content of the URI to categorize the page into one of the above five types (article, discussion, etc.), and the returned data object is added to the set of selected objects for the question. The top ranked tag associated with the object is always harvested. For the remaining tags, however, they are harvested only if they clear the following two-step filtering approach. The first filter is based on the confidence score associated with the tag. Specifically, for object X, its tags with confidence score higher than the average confidence score, computed on tags for object X, are selected. From this subset of tags, only the ones which match at least one n-gram from the set G are retained. The resulting set of tags is harvested and the additional content obtained by each tag becomes part of the original data object associated with that tag.

*2) **Data Object Selection**:* The goal of this component is to select the best data objects from all the objects selected in the previous step. To accomplish this, first the data objects selected for each query are ranked based on the $S(o)$ scores. Thus, for each query, q_i, an ordered set of data objects, $O_i = (o_i^1, ..., o_i^k)$ is now available. Each set O_i is locally sorted, but the global set of data objects $O_1, ..., O_m$ retrieved for queries $q_1, ..., q_m$, is not sorted. In order to identify the n best objects, our system uses the following global scoring function to rank the data objects. Let O be the set of unique objects derived from $O_1, ..., O_m$. $\forall o \in O, GS(o) = (\sum_{i=1}^{i=m} \frac{1}{rank(o, O_i)})$, where $rank(x, y)$ returns the rank position of the object x in the ordered set y. This function captures both the rank position, and retrieval frequency of each object. Thus, objects that are ranked highly in more number of individual ordered sets will be scored higher using this function. We wish to retain only a few such highly ranked objects. For this final selection step we employ the same approach used for query-level top object selection in Section II-C1. However in this case, the global score, $GS(o)$, is used instead of the object scoring function $S(o)$. The selected objects are then passed on to the next module as the most promising answer-bearing objects. In case of ties, the object with the most recent time-stamp is ranked first, to favor more current data. In our running example, the highest score is tied between two objects. However, the one with earlier time-stamp specifies 20,320 feet as the height of Mount McKinley, but the one with more recent time-stamp states that new geological surveys announced that Mount McKinley is 20,310 feet high instead.

D. Answer Extraction and Selection Module (AESM)

This module is tasked with extracting candidate answers from the objects selected by the previous module, and then selecting the best sentences to construct the final answer.

1) Candidate Answer-sentence Extraction: This module starts by extracting data from the textual fields of the various objects selected by KGOEM. The text data is then segmented at sentence level, and then vectorized using tf-idf representation. For each of these sentences s_i, a score $sent\_score(s_i)$ is assigned as follows:

$$I\Delta_{dist}(s_i) = (\frac{|P| - 1}{\sum_{j=1}^{|P|-1}(p_{j+1} - p_j)})$$

$$sent\_score(s_i) = \cos(\vec{s_i}, \vec{s_0}) + I\Delta_{dist}(s_i)$$

where $\cos(\vec{s_i}, \vec{s_0})$ function computes the cosine similarity between the sentence s_i vector and the question vector s_0. The $I\Delta_{dist}(s_i)$ function is the Inverse Delta Distance, where p_j is position offset of a term that appears in both, question and the answer-sentence, s_i. P is an ordered set of offsets for such matching terms in s_i. In essence, $I\Delta_{dist}(s_i)$ computes the inverse of the average distance between matching terms (in words). The intuition behind this is that positional affinity of question terms in the answer is an indicator of answer relevance. For our running example "How tall is Mount McKinley?" below are two candidate answer-sentences. The second sentence is scored higher by $sent\_score()$ than the first due to higher cosine similarity score (more terms overlap between sentence and question), and also higher inverse delta distance (shorter distance between matching terms "*mount*", "*McKinley*" and "*tallest*")

- "The administration decided to change the name of mount McKinley to Denali, even though it had been known as McKinley for decades"
- "If the 20,237 feet calculation is correct, that makes mount McKinley the tallest peak, still, by more than 680 feet"

2) Answer Selection: Now that the candidate answer-sentences are scored, the average score over all the sentences is computed, and only the top sentences with higher than or equal to average score are retained for this step. These sentences are then grouped based on their source *sections*, such as, the different posts in a discussion object type, to generate the final answer. This simple approach offers two advantages. First, the length of the generated answer (in terms of number of sentences) is data-driven. Depending upon the distribution of the computed scores, the number of answer-sentences with higher than average can be few or more. In the extreme case of uniform distribution of scores, all the answer-sentences will be retained to generate the final answer since the score of every sentence will be equal to the average score. Second, grouping the sentences based on their source section improves the flow of the generated answer.

III. Experimental Methodology

The experimental methodology adopted to empirically test the proposed approach is described next. We assess the efficacy of the approach at answering both, factoid, and non-factoid questions.

A. Baseline QA Systems

We employ four baseline systems to conduct a thorough comparative analysis. The first two baselines (BKGQA, SBKGQA) are variants of our approach that illustrate the utility of different sub-components. The BKGQA system uses the question directly as a single query to the Diffbot knowledge graph, the top-ranked single data object in the retrieved results is selected as the answer-bearing object. Three sentences with highest cosine similarity with the question are concatenated to generate the final answer. The SBKGQA system performs multi-query search, however, its formulated queries are not restricted to the text and title fields. Instead of the object scoring function $S(o)$ described in section II-C1, only the confidence score provided by Diffbot is used to rank the objects for each query. Lastly, the answer selection approach is the same. The other two baselines are state-of-the-art QA systems as ranked by the TREC LiveQA task. SF-State-QA [5] is a QA system that uses sophisticated query generation approach based on dependency parsing, and grammatical rules. The query is then run against the Web using commercial search engines' APIs, to obtain up-to-date and high-quality web-pages. Finally, the answers extracted from these web-pages are ranked using trained answer ranking models. Although this system does not use a knowledge graph as its information source, it is a well-designed QA system that harnesses the information on the Web through the commercial search engine APIs. The Open Advancement of Question Answering (OAQA) [29], and the Encoder-Decoder [30] are also two baselines that we use in our evaluation results. The first one outperformed all its competitors in TREC 2015 LiveQA Track [10], and the latter is an improved version of OAQA presented in the TREC 2016 LiveQA Track [9]. OAQA uses a BLSTM to process the data while Encoder-Decoder uses a Recurrent Neural Network model. We also include as baseline EmoryCrowed as a reference baseline since it outperformed all the other systems in [9].

B. Evaluation Data and Metrics

A set of 100 factoid questions was sampled at random from an open source benchmarking dataset, which consists in a combination of two sub-datasets, IRC and TREC of 867 curated factoid questions (v2) [13], For evaluation with non-factoid questions, we sampled a set of 100 questions each from TREC LiveQA 2015 [15], and 2016 [16] datasets, each of which contains 1000+ non-factoid and non-curated questions. To evaluate the end-to-end performance of our system, we conducted manual assessment of the answers generated by our system for the sampled questions. Each question-answer pair was judged by two annotators on the official scale used at TREC LiveQA 2015 (0: non-readable or unanswered, 1: poor, 2: fair, 3: good, 4: excellent). The complete set of question-answers from TREC LiveQA 2015 (1087 Questions), and 2016 (1015 Questions) is used for automated evaluation described in Section IV-A. **Effectiveness metrics:** To quantify the overall effectiveness of the QA systems we employ the official metrics established by TREC LiveQA: avgScore(0-3), and success@i+ metrics. Recall that every answer is assigned a score between 0 and 4 by annotators. The scores 1 through 4 are adjusted to 0 through 3 scale, first. Then, the average score over all the answers is computed to generate the avgScore(0-3) value. The score adjustment ensures that poor answers (1) do not contribute to the avgScore value. succ@i+ metrics is the ratio of number of questions with scale i or above, and the toal number of questions. In addition, we employ Cosine similarity, and Jaccard coefficient to algorithmically evaluate the answer quality by comparing it with human generated answers using these similarity metrics. **Efficiency metrics:** The efficiency is quantified using average run-times for each individual module, and for the end-to-end system.

TABLE I
End-to-end QA system evaluation results.

	avgScore(0-3)	succ@2+	succ@3+	succ@4+
TREC 2015 non-factoid				
BKGQA	0.520	0.360	0.130	0.030
SBKGQA	1.480	0.650	0.480	0.356
KGQA	1.920	0.880	0.640	0.400
SF-State-QA	1.420	0.650	0.430	0.340
OAQA	1.081	0.532	0.359	0.190
TREC 2016 non-factoid				
BKGQA	0.400	0.290	0.090	0.020
SBKGQA	1.670	0.730	0.540	0.400
KGQA	1.960	0.850	0.650	0.460
SF-State-QA	1.570	0.760	0.500	0.330
EmoryCrowed	1.260	0.620	0.421	0.220
Encoder-Decoder	1.154	0.560	0.395	0.199
IRC-TREC Curated factoid				
BKGQA	0.310	0.231	0.057	0.035
SBKGQA	1.690	0.800	0.540	0.350
KGQA	2.170	0.920	0.690	0.560

IV. Results and Analysis

We evaluated the developed system's performance, in terms of effectiveness, and efficiency, both. The results and their analyses are reported in this section.

A. Answers Quality

Table I reports the results for the non-factoid and factoid questions datasets.

Compared to two of the baseline systems, BKGQA and SBKGQA, the proposed system, KGQA performs consistently and substantially better across all the metrics. We see this trend for all the datasets, TREC LiveQA 2015, TREC LiveQA 2016 and IRC-TREC. Recall that BKGQA and SBKGQA also use the same information source, Diffbot KG, and thus this difference in performance is especially noteworthy. It illustrates the value of two key aspects of our approach: 1. multi-query formulation targeting specific fields, and 2. tag harvesting. The former facilitates manifold probing of the KG for the same question, and the latter exploits the underlying graph to assimilate information that is indirectly related to the question.

One of the key challenges for researchers working on question answering systems, especially with non-factoid questions, is evaluation. One, it is nearly impossible to automate the answer evaluation task. As such, we conducted manual assessment for both factoid and non-factoid datasets, as is described in Section III-B. The second challenge is not specific to QA systems – reproducing results from other QA systems. The evaluation methodology used by Pithyaachariyakul et al., 2018 [5] for the SF-State-QA system is the most comparable to ours, where 100 randomly selected question-answer pairs were manual assessed for quality. As per the results in Table I, when comparing KGQA with SF-State-QA, the trends are inconsistent across the two datasets for succ@2+ metrics. However, if the focus is on high-quality answers (good and excellent), then a consistent trend emerges, KGQA outperforms SF-State-QA for both datasets. Table I also specifies the performance of two other QA systems, OAQA [29], and Encoder-Decoder [30], that were the best performing systems in the LiveQA track at TREC 2015, and TREC 2016, respectively. The results for these systems are computed over the complete question collection of TREC 2015 and 2016 datasets (1000+ questions each). As such, these result numbers are not directly comparable to the results for KGQA. However, since KGQA results are computed on a subset of these question collections, we can gather indicative trends. The KGQA system outperforms the state-of-the-art systems from both years. The improvements are bigger for high-quality answers, which is in-line with the trend observed with the other baseline system.

Table II provides results for a larger set of questions that employs overlap-based automated evaluation. Since this evaluation does not use explicit relevance judgments for the generated answers, the absolute result values are less important. Instead, we analyze the relative differences in the values reported in Table II. The questions are categorized based on their length (in words) in order to investigate the effects of query length on the system performance. We see substantial and consistent improvement in performance with KGQA for short questions, and for long questions. For medium length questions, all three systems are comparable when performance is quantified with cosine similarity. Table II also reports the average number of objects retrieved from KG per question. Recall that for the baselines, top 10 objects are retrieved for each question. KGQA however adopts a dynamic strategy where the number of selected objects is dependent on the question. These numbers demonstrate the value of employing this dynamic strategy. For 2015 questions, only a few objects are needed, while for 2016 questions, lot more than top 10 objects are needed to provided the competitive effectiveness. The last set of columns in Table II report the number of unanswered questions for each system. KGQA is consistently more robust than the other systems.

B. System Efficiency

For all three datasets, Table III reports the average runtime per question for the end-to-end system, and for every individual module. As is evidenced by these results, KGQA generates an answer for a non-factoid question in 32.5 seconds, and for factoid questions in 20.1 seconds, on average. This response time is substantially better than both, SF-State-QA, and BKGQA. The response times are not made available by

TABLE II
Overlap-based evaluation results for KGQA system and baseline systems.

Metric	**Jaccard Sim**			**Cosine Sim**			**Avg #Objects Ret**			**#Unanswered Questions**		
Questions tokens	**[0;10]**	**[11;32]**	**[33;)**	**[0;10]**	**[11;32]**	**[33;)**	**[0;10]**	**[11;32]**	**[33;)**	**[0;10]**	**[11;32]**	**[33;)**
TREC 2015 non-factoid												
Num Questions	64	404	619	64	404	619	64	404	619	64	404	619
BKGQA	0.058	0.061	0.054	0.288	0.307	0.297	10	10	10	3	61	41
SBKGQA	0.089	0.067	0.077	0.291	0.272	0.313	10	10	10	2	42	18
KGQA	0.143	0.163	0.166	0.336	0.298	0.326	9	3	2	3	27	12
TREC 2016 non-factoid												
Num Questions	122	343	550	122	343	550	122	343	550	122	343	550
BKGQA	0.043	0.063	0.066	0.206	0.292	0.302	10	10	10	3	47	45
SBKGQA	0.066	0.055	0.068	0.368	0.307	0.324	10	10	10	13	33	30
KGQA	0.184	0.166	0.176	0.386	0.292	0.335	16	17	16	11	22	37

TABLE III
AVERAGE RUNTIME PER QUESTION (MILLISECONDS)

	End-to -End	Query Formulation	Object Extraction	Answer Selection
Non-Factoid TREC 2015-2016				
BKGQA	35297.83	1.82	0.34	35295.67
SBKGQA	32629.14	1.65	0.17	32627.32
KGQA	32549.12	1.67	0.23	32547.22
SF-State-QA	36700.00	410.00	4360.00	41470.00
Factoid IRC-TREC Curated Wikipedia Based				
BKGQA	22512.16	1.51	0.09	22510.56
SBKGQA	20495.32	1.31	0.10	20493.90
KGQA	20145.07	1.22	0.07	20143.78

the other baseline systems, OAQA, and Encoder-Decoder, and thus the efficiency of these baselines cannot be analyzed.

At the module-level, one would expect BKGQA to be faster than KGQA in the query formulation module, since BKGQA simply uses the question as a query. However, that is not the case because stop-words are retained in the BKGQA approach which make KG querying slower. SBKGQA and KGQA have similar performance in the query formulation module. However, during the object extraction procedure SBKGQA is slightly faster than KGQA because the latest usually processes more objects for that module. SF-State-QA employs a relatively shallow linguistic approach to transform the question to query, but even this light-weight approach has a high latency. This step is nearly 400x slower for that approach.

The shortest runtime per question reported by KGQA is for the object extraction module. This indicates that the additional probing of the KG with the harvested tags does not add unreasonable overhead to this module. Furthermore, the data fusion step employed to select the single best object also is very efficient. The slowest module for all the three QA systems is the answer selection module, which is expected, since large amounts of textual data is being processed and vectorized in this step.

V. CONCLUSIONS

This paper presented KGQA, a framework for knowledge graph based question answering approaches that can respond to factoid and non-factoid questions from any topic domain. This framework harnesses the salient features of the rich and comprehensive KGs learned from the WWW. Although we test the framework with a specific KG in this paper, the framework is designed to facilitate easy adaptation to other KGs. An end-to-end evaluation with multiple datasets and baselines demonstrates that the proposed approach consistently outperforms all the baselines. This is especially true for high-quality answers. In terms of efficiency as well, the proposed approach provides the fastest response time. Testing the framework with other KGs, and designing approaches to jointly use multiple KGs to improve the overall coverage are part of the future work.

REFERENCES

[1] Malik M. K. Rashid M. U. Zafar R. Abbas, F. Wikiqa a question answering system on wikipedia using freebase, dbpedia and infobox. Sixth International Conference on INTECH, 2016.

[2] Eugene Agichtein, David Carmel, Dan Pelleg, Yuval Pinter, and Donna Harman. Overview of the trec 2015 liveqa track. In *TREC*, 2015.

[3] Fischer S Black. *A deductive question answering system*. Harvard University, 1964.

[4] Evans C. Paritosh P. Sturge T. Bollacker, K. and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD, 2008.

[5] Pithyaachariyakul C. and Kulkarni A. Automated question answering system for community-based questions. In proceedings of the Thirty-Second AAAI New Orleans, USA., 2018.

[6] Agarwal K. Purohit S. Zhang B. Pirrung M. Smith W. Thomas M. Choudhury, S. Construction and querying of dynamic knowledge graphs. 2017 IEEE 33rd ICDE, 2016.

[7] L Stephen Coles. An on-line question-answering systems with natural language and pictorial input. In *Proceedings of the 1968 23rd ACM*, pages 157–167. ACM, 1968.

[8] J. Lehmann S.Auer D. Lukovnikov, A Fisher. Neural network-based question answering over knowledge graphs on word and character level. WWW '17 Proceedings of the 26th International Conference on World Wide Web., 2017.

[9] Dan Pelleg Yuval Pinter Donna Harman Eugene Agichtein, David Carmel. Overview of the trec 2016 liveqa track. 2016.

[10] Donna Harman Dan Pelleg Yuval Pinter2 Eugene Agichtein, David Carmel. Overview of the trec 2015 liveqa track. 2015.

[11] Ben Hixon, Peter Clark, and Hannaneh Hajishirzi. Learning knowledge graphs for question answering through conversational dialog. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 851–861, 2015.

[12] http://rtw.ml.cmu.edu/rtw/kbbrowser/.

[13] https://github.com/brmson/dataset-factoid curated.

[14] https://pypi.python.org/pypi/grammar check/1.3.1.

[15] https://trec.nist.gov/data/qa/2015_LiveQA/questions.txt.

[16] https://trec.nist.gov/data/qa/2016_LiveQA/questions.txt.

[17] https://www.diffbot.com/dev/docs/article/.

[18] https://www.diffbot.com/knowledge graph/.

[19] https://www.google.kg.

[20] https://www.nltk.org.

[21] Zou L. Wang H. Zhao D Hu, S. Answering natural language questions by subgraph matching over knowledge graphs. IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING., 2017.

[22] Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. Evaluating question answering over linked data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 21:3–13, 2013.

[23] Dan Moldovan, Marius Paşca, Sanda Harabagiu, and Mihai Surdeanu. Performance issues and error analysis in an open-domain question answering system. *ACM Transactions on Information Systems (TOIS)*, 21(2):133–154, 2003.

[24] John Prager et al. Open-domain question–answering. *Foundations and Trends® in Information Retrieval*, 1(2):91–231, 2007.

[25] Peter S Rosenbaum. A grammar base question-answering procedure. *Communications of the ACM*, 10(10):630–635, 1967.

[26] Pum-Mo Ryu, Myung-Gil Jang, and Hyun-Ki Kim. Open domain question answering using wikipedia-based knowledge model. *Information Processing & Management*, 50(5):683–692, 2014.

[27] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[28] Robert F Simmons. Natural language question-answering systems: 1969. *Communications of the ACM*, 13(1):15–30, 1970.

[29] Di Wang and Eric Nyberg 2015. Discovering the right answer with clues. 2015.

[30] Di Wang and Eric Nyberg 2016. An attentional neural encoder-decoder approach for answer ranking. 2016.

[31] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[32] Xiaodong He Wen-tau Yih, Ming-Wei Chang and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, 2015.

[33] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 956–966, 2014.