CBench: Demonstrating Comprehensive Evaluation of Question Answering Systems over Knowledge Graphs Through Deep Analysis of Benchmarks

Abdelghny Orogat
Carleton University
abdelghny.orogat@carleton.ca

Ahmed El-Roby
Carleton University
ahmed.elroby@carleton.ca

ABSTRACT

A plethora of question answering (QA) systems that retrieve answers to natural language questions from knowledge graphs have been developed in recent years. However, choosing a benchmark to accurately assess the quality of a question answering system is a challenging task due to the high degree of variations among the available benchmarks with respect to their fine-grained properties.

In this demonstration, we introduce CBench, an extensible, and more informative benchmarking suite for analyzing benchmarks and evaluating QA systems. CBench can be used to analyze existing benchmarks with respect to several fine-grained linguistic, syntactic, and structural properties of the questions and queries in the benchmarks. Moreover, CBench can be used to facilitate the evaluation of QA systems using a set of popular benchmarks that can be augmented with other user-provided benchmarks. CBench not only evaluates a QA system based on popular single-number metrics but also gives a detailed analysis of the linguistic, syntactic, and structural properties of answered and unanswered questions to help the developers of QA systems to better understand where their system excels and where it struggles.

PVLDB Reference Format:
Abdelghny Orogat and Ahmed El-Roby. CBench: Demonstrating Comprehensive Evaluation of Question Answering Systems over Knowledge Graphs Through Deep Analysis of Benchmarks. PVLDB, 14(12): 2711 - 2714, 2021.
doi:10.14778/3476311.3476326

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/aorogat/CBench.

1 INTRODUCTION

Recent years witnessed unprecedented growth in the number of knowledge graphs (KGs). As a result, a large number of QA systems that let users describe their information needs using natural language were developed. In fact, over 62 QA systems have been developed since 2010 [2]. To evaluate these QA systems, several benchmarks were introduced (e.g., [4–6]). These benchmarks typically include questions described in natural language, answers to the questions from the KG targeted by the benchmark, and possibly structured queries that return the previously mentioned answers. To evaluate a newly developed QA system, its developers need to choose from a large number of benchmarks (at least 17 at the time of writing this paper) to evaluate their system. Without a quantitative comparison that highlights the differences between these benchmarks, choosing a subset of them to evaluate a new QA system is mainly motivated by the ease of comparison to existing systems in the literature rather than by how effective a benchmark is in evaluating a QA system.

In this demonstration, we showcase our comprehensive benchmarking suite (CBench [3]) that can be used to deeply analyze QA benchmarks with respect to several linguistic, syntactic, and structural properties. Using this fine-grained analysis, we surprisingly reveal that the benchmarks that are widely used in the literature vary significantly with respect to these properties, which in turn affects the assessment of QA systems. That is, depending on the benchmark used to evaluate a QA system, it can be shown to either outperform or underperform another QA system. Therefore, to facilitate a more comprehensive comparison between QA systems, CBench can be also used to evaluate QA systems using any of the prepackaged benchmarks that can be augmented with any user-provided benchmark and provide the user with a detailed interactive report that includes not only the quality scores of the QA system, but also details of the properties of the correctly and incorrectly answered questions/queries, which in turn can help the QA system developers to better understand the strengths and weaknesses of their system. CBench can be used in two modes: The *Benchmark Analysis* and *QA Evaluation* modes.

Benchmark Analysis Mode: CBench is used in this mode to analyze the benchmarks chosen by the user with respect to several syntactical and structural properties of the queries in the benchmark, and linguistic properties of the natural language questions. Specifically, CBench analyzes the query keywords, the number of triple patterns, the query operators (syntactical properties), and the different query shapes (structural properties). For the natural language questions, CBench analyzes the question type, the length of the question, the part-of-speech (PoS) tags, and the dependency parse tree (linguistic properties). Using this mode, we reveal that the benchmarks used in the literature vary significantly with respect to these properties, which affects the reported quality scores of the QA systems (can be obtained using the QA Evaluation mode).

QA Evaluation Mode: CBench is used in this mode to facilitate the comparison of QA systems. Traditionally, benchmarks are used in the following fashion: The user parses the benchmark file, extracts the questions and utilizes the QA system to find answers in the targeted KG, then compare the returned answers to the answers

Proceedings of the VLDB Endowment, Vol. 14, No. 12 ISSN 2150-8097.
doi:10.14778/3476311.3476326

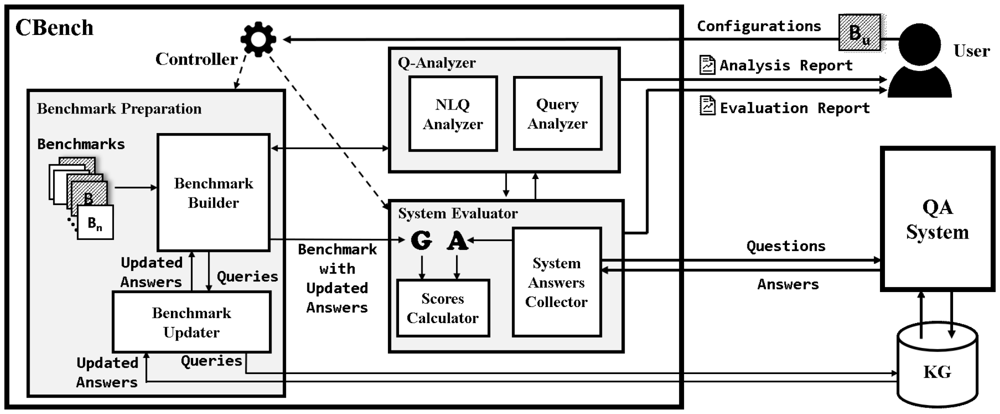


Figure 1: The architecture of CBench.

extracted from the benchmark file to calculate multiple evaluation scores like micro, macro, and global F-1 scores. The user then examines the questions that the QA system failed to answer correctly and debug their code to identify why the QA system struggles with these questions. In that sense, the benchmark is used as a dataset that helps in producing the aforementioned scores with a lost potential of being more informative to its users by giving more details on the fine-grained properties of the processed questions, which will help the user better understand how the QA system behaves. CBench overcomes this usability problem by facilitating the evaluation through prepackaging all the benchmarks that are used in the literature within CBench and supporting multiple APIs for interaction with different QA system architectures and providing in-depth insights on how the QA systems perform beyond the single-number scores. In addition to reporting micro, macro, and global F-1 scores, CBench also analyzes all the questions in the chosen benchmarks and their corresponding structured queries (if available). Specifically, CBench returns (1) a detailed analysis of the properties of the queries that the evaluated QA system processed, and (2) linguistically-similar natural language questions to any question of interest (e.g., a question that the QA system failed to answer). Using the two aforementioned types of output, the QA system developers can either (1) identify common properties between questions that the QA system struggles with (e.g., most of the questions have a specific query shape), or (2) identify obvious inconsistencies in the processed questions. For example, using CBench, we were able to quickly identify that one of the QA systems we evaluated was able to answer the question “What is the capital of Cameroon?” correctly, while it incorrectly answered “What is the capital of Canada?”, which highlights an overfitting problem in its entity recognition and relation mapping approaches. Being able to quickly identify commonalities or inconsistencies will help the QA system developers to quickly identify the QA system component that they need to improve. Based on the insights provided by CBench, the user can also use it in a *Debugging Mode* within the QA Evaluation Mode, where they can control CBench’s output questions based on any of the linguistic, syntactical, or structural properties of all the questions and queries in CBench to better understand how their QA system behaves in several controlled situations. For example, the user can choose to evaluate their QA system only on aggregate questions (e.g., How many) whose queries have a star-shape to investigate how their system processes such questions.

2 OVERVIEW OF CBENCH

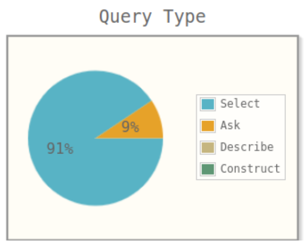
In this section, we discuss (1) the architecture of CBench, (2) the types of analysis performed over the structured SPARQL queries and the natural language questions of the prepackaged or user-provided benchmarks, and (3) the evaluation features of CBench.

2.1 CBench Architecture

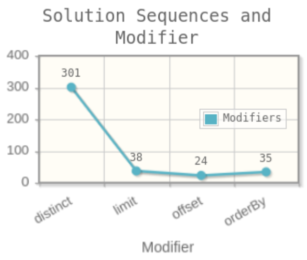
Figure 1 shows the architecture of CBench, which can be used in two modes: (1) The Benchmark Analysis Mode, where CBench can be used to perform a fine-grained analysis on the structured queries and the natural language questions on a set of benchmarks selected by the user, and (2) the QA Evaluation Mode, where CBench can be used to evaluate QA systems over the user-selected benchmarks providing deeper insights on how the QA systems are performing.

Benchmark Analysis Mode: CBench includes 17 benchmarks from which the user can choose a subset for analysis. The user can also upload their own benchmarks to be included in the analysis. The Benchmark Builder passes the selected benchmarks (and the uploaded ones, if any) to the Q-Analyzer which carries out the syntactical and structural analysis of the queries (Section 2.2), and the linguistic analysis (Section 2.3). Finally, the Q-Analyzer returns the analysis report to the user.

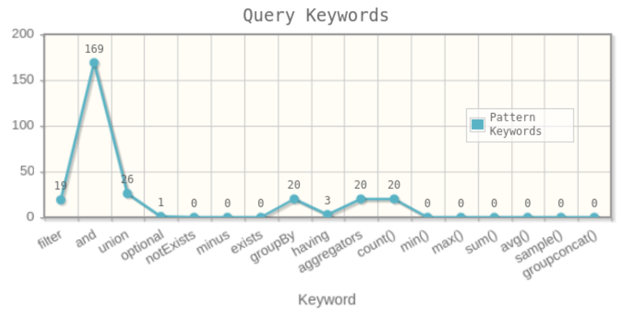
QA Evaluation Mode: Just like the previous mode, the user selects a set of benchmarks and/or uploads their own to evaluate the QA system. In addition, the user provides CBench with a URL for an endpoint that CBench can query. Other configuration parameters that are used in the evaluation (e.g., thresholds for calculating quality scores) are also chosen by the user prior to evaluation. To avoid the scenario where the selected benchmarks target different versions of the same KG, the Benchmark Builder updates the answers of the queries in the selected benchmarks through the Benchmark Updater module, which queries the used KG that will be used for the evaluation to retrieve the updated answers. The updated benchmarks are then passed to the System Evaluator. The System Evaluator carries out three tasks: (1) Communicating with the QA system to collect the answers to the questions from the selected benchmarks, (2) calculating the micro, macro, and global F-1 scores (discussed in Section 2.4), and (3) retrieving the fine-grained analysis of the processed questions from the Q-Analyzer.



(a)



(b)



(c)

Figure 2: Analysis of the query keywords for QALD-9.

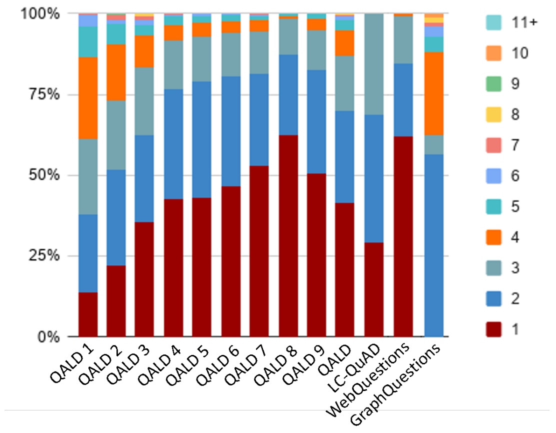


Figure 3: Percentage of queries exhibiting different number of triple patterns for each benchmark.

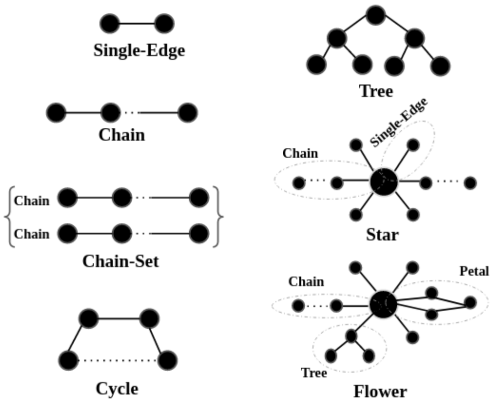


Figure 4: The different shapes recognized by CBench

The System Evaluator then outputs an interactive report that includes the scores and the analysis of the processed questions to the user. The user can choose to focus on specific questions to view all their fine-grained properties. CBench also finds other questions that are linguistically-similar to the selected questions (discussed in Section 2.3). Using this feature, the user is able to quickly identify either common features or inconsistencies of unanswered or incorrectly answered questions, which will help the user to understand which components of their QA system to improve. The user can also use CBench in the Debugging Mode, in which they can group questions/queries based on specific properties to evaluate their QA system in specific scenarios.

The details of the configurations of CBench, how to add a new benchmark, and the APIs used for communication with the QA systems can be found in the system's repository[1].

2.2 Analysis of Structured Queries

2.2.1 Syntactical Analysis. This type of analysis focuses on the syntactical properties of the SPARQL [1] queries in the benchmarks. In CBench, we focus on three properties: (1) The query keywords, (2) the size of the queries represented by the number of triple patterns in the query, and (3) the combination of operators in the queries.

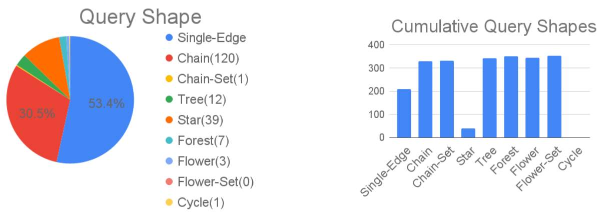
Query Keywords: This type of analysis focuses on the frequency of the keywords of the SPARQL queries in the benchmarks. The keywords existing in the benchmarks are mainly used as solution modifiers, query operators, or aggregates. Figure 2 shows an example of the output of CBench for the QALD-9 benchmark [6]. Figure 2a shows the distribution of the keywords specifying the type of the query. The figure shows that most of the queries use the *Select* keyword, and only 9% of the queries are yes/no questions that are described in SPARQL using the *Ask* keyword. Figure 2b shows the frequency of the solution modifier keywords (excluding *Select*), and Figure 2c shows the frequency of the remaining keywords identified by CBench.

Number of Triple Patterns: This type of analysis highlights the length of the queries in the benchmarks represented as the number of triple patterns in each query. Figure 3 shows an example of the output for all the prepackaged benchmarks that include SPARQL queries in CBench.

Query Operators: In CBench, we focus on the following operators: conjunction of triple patterns, filtering of output, including optional graph patterns, and union of graph patterns. The queries in the benchmarks may include none or combinations of these operators. CBench shows the percentage of the different combinations of these operators for all the queries.

2.2.2 Structural Analysis. In addition to the syntactical analysis of the queries in the benchmarks, CBench also studies the structural shapes of the queries. CBench identifies eight different shapes of queries. Figure 4 illustrate these shapes. Figure 5 shows output of the structural analysis for QALD-9. As there are shapes that subsumes other shapes (e.g., the star shape subsumes the single-edge and the chain shapes), CBench provides two figures for the shapes frequencies. While Figure 5a shows the percentages of the shapes ignoring the shapes they subsume (e.g., Forest shapes that

[1]https://github.com/aorogat/CBench



(a) (b)

Figure 5: Analysis of the shapes of queries in QALD-9.

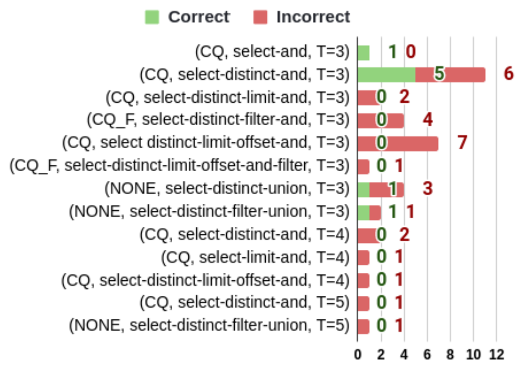


Figure 6: The properties of the Star-shaped queries of the questions from QALD-9 answered by WDAqua. CQ means conjunctive queries, CQ_F means Conjunctive queries with filtering, NONE means no conjunction of triple patterns.

are not tree shapes), Figure 5b shows the cumulative result (e.g., Flower-Set subsuming all other shapes).

2.3 Analysis of Natural Language Questions

In CBench, we also analyze the natural language questions via the NLQ Analyzer to provide linguistic-based insights on the questions in the benchmarks. The analysis of the natural language questions in CBench focuses on the type of the natural language questions (e.g., Wh-questions, aggregate, temporal, yes/no, requests, etc.) and the length of the question in terms of the count of tokens of the question.

To support retrieving linguistically-similar questions to any question chosen by the user when evaluating a QA system (discussed next), CBench converts the natural language questions into their corresponding vector space using our custom embedding function that uses a Part-of-Speech (PoS) tagger to tag each token in the question, then updates the frequency of the tag in the final vector representation, which has a number of dimensions that is equal to the number of possible tags. This representation not only captures the definition and the context of each token but also indirectly captures the length of the question by counting the frequencies of the occurrences of the tags. We use the obtained vector representation to calculate how linguistically similar a pair of questions are. In CBench, we use the euclidean distance measure to represent how dissimilar two questions are. We also apply a similar approach using a dependency parse tree replacing the PoS tagger in the embedding function with no significant change of the reported similar questions.

2.4 Evaluation of QA Systems

CBench can be used to evaluate multiple QA systems that are running either locally, or remotely and accessed via web services. The user may choose to evaluate the QA systems using any subset of the benchmarks prepackaged in CBench, or upload their own benchmark. CBench evaluates the QA systems beyond the single-number scores that other evaluation platforms offer (e.g., Gerbil [7]). In addition to reporting the traditional quality scores (micro F-1, macro F-1, and global F-1), the user can use CBench to group sets of questions/queries by the aforementioned properties or choose to evaluate the QA systems in a debugging mode, where the user has full control on which questions/queries are used in the evaluation. The user can choose to evaluate the QA systems using a subset of questions/queries that have specific properties. For example, the user may be interested in aggregate questions whose queries have a tree shape. Figure 6 shows the output shown to the user for answering star-shaped queries from QALD-9 using WDAqua.

3 DEMONSTRATION SCENARIO

We prepackage CBench with all the benchmarks available in the literature of QA over KGs and link six different QA systems to CBench, three of which run locally and three run remotely to demonstrate the flexibility and ease-of-use of CBench. First, the participants will use CBench's Benchmark Analysis Mode to witness the high-degree variations across all the benchmarks with respect to the fine-grained properties of the queries/questions in the benchmarks. The variations will be shown to be statistically significant. Then, to demonstrate how these variations affect the assessment of QA systems, the participants will be able to evaluate the six QA systems using subsets of the benchmarks and witness that by changing the benchmark used and the quality score metric, the rankings of the QA systems change. Finally, to demonstrate how effective CBench is in giving insights to the QA system developers, the participants can navigate through the interactive evaluation report to view the fine-grained properties of questions/queries that are correctly/incorrectly answered, and linguistically similar questions to any question of interest. Through this interaction, the participants will be able to gain insights into the weaknesses of the QA systems without having to read the details of how the systems were designed or examine their source codes.

REFERENCES

[1] SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/, 2013.
[2] K. Höffner, S. Walter, E. Marx, R. Usbeck, J. Lehmann, and A.-C. Ngonga Ngomo. Survey on challenges of question answering in the semantic web. *Semantic Web*, 8(6), 2017.
[3] A. Orogat, I. Liu, and A. El-Roby. CBench: Towards better evaluation of question answering over knowledge graphs. *Proceedings of the VLDB Endowment (PVLDB)*, 14(8), 2021.
[4] Y. Su, H. Sun, B. Sadler, M. Srivatsa, I. Gur, Z. Yan, and X. Yan. On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.
[5] P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference (ISWC)*, 2017.
[6] R. Usbeck, R. H. Gusmita, M. Saleem, and A.-C. N. Ngomo. 9th challenge on question answering over linked data (QALD-9). *Joint Workshop on Natural Language Interfaces for Web of Data (NLIWoD) and Question Answering over Linked Data challenge*, 2018.
[7] R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A.-C. Ngonga-Ngomo, C. Demmler, and C. Unger. Benchmarking question answering systems. *Semantic Web*, 10(2), 2019.