BigText-QA: Question Answering over a Large-Scale Hybrid Knowledge Graph

Jingjing Xu[1(✉)], Maria Biryukov[1], Martin Theobald[1], and Vinu Ellampallil Venugopal[2]

[1] University of Luxembourg, 4365 Esch-sur-Alzette, Luxembourg
{jingjing.xu,maria.biryukov,martin.theobald}@uni.lu
[2] International Institute of Information Technology (IIIT), Bangalore, India
vinu.ev@iiitb.ac.in

Abstract. Answering complex questions over textual resources remains a challenge, particularly when dealing with nuanced relationships between multiple entities expressed within natural-language sentences. To this end, curated knowledge bases (KBs) like YAGO, DBpedia, Freebase, and Wikidata have been widely used and gained great acceptance for question-answering (QA) applications in the past decade. While these KBs offer a structured knowledge representation, they lack the contextual diversity found in natural-language sources. To address this limitation, BigText-QA introduces an integrated QA approach, which is able to answer questions based on a more redundant form of a knowledge graph (KG) that organizes both structured and unstructured (i.e., "hybrid") knowledge in a unified graphical representation. Thereby, BigText-QA is able to combine the best of both worlds—a *canonical set of named entities*, mapped to a structured background KB (such as YAGO or Wikidata), as well as an *open set of textual clauses* providing highly diversified relational paraphrases with rich context information. Our experimental results demonstrate that BigText-QA outperforms DrQA, a neural-network-based QA system, and achieves competitive results to QUEST, a graph-based unsupervised QA system.

Keywords: Question Answering · Large-Scale Graph · Hybrid Knowledge Graph · Natural Language Processing

1 Introduction

Information extraction (IE) has made strides in extracting structured data ("facts") from unstructured resources like text and semistructured components (e.g., tables and infoboxes) [49]. Established knowledge bases (KBs) such as YAGO [44], DBpedia [2], Freebase [6] or Wikidata [47] use IE techniques to store numerous facts. However, KBs are mostly limited to triple-based representations of knowledge, capturing semantic relationships between real-world objects (entities and concepts) but lacking contextual information about the facts' origins. On the other hand, information retrieval (IR) efficiently

Supported by Luxembourg National Research Fund (FNR).

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2024
Published by Springer Nature Switzerland AG 2024. All Rights Reserved
Z. Tan et al. (Eds.): BDTA 2023, LNICST 555, pp. 33–48, 2024.
https://doi.org/10.1007/978-3-031-52265-9_3

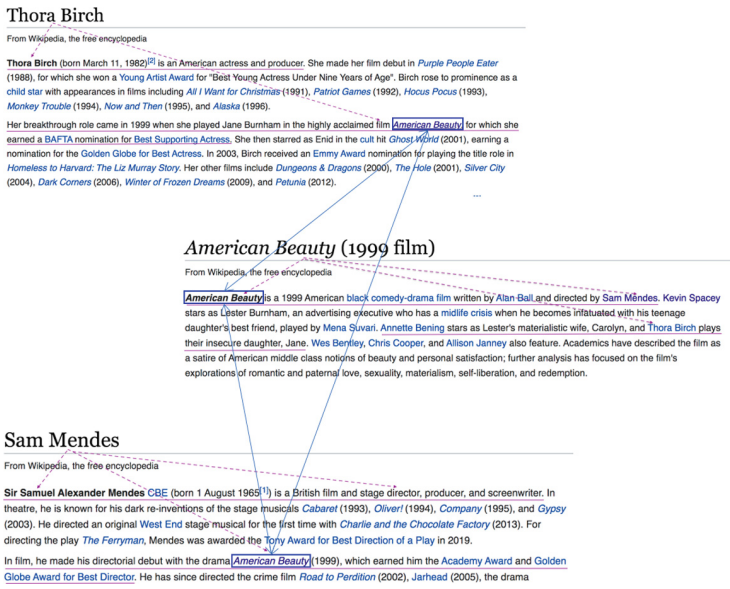

Fig. 1. A snapshot of the BigText graph viewed as a corpus of interconnected documents.

operates on large document collections using context-based statistics like term frequencies and co-occurrences [34]. Yet, classical IR approaches often rely on a simplified "bag of words" representation, overlooking the documents' internal structure.

Our BigText approach aims to combine the strengths of information extraction (IE) and information retrieval (IR) without simplifying their concepts. It represents a document collection as a redundant (hybrid) knowledge graph (KG), preserving the original document structure, domains, hyperlinks, and metadata. The graph includes substructures like sentences, clauses, lists, and tables, along with mentions of named entities and their syntactic and semantic dependencies. By linking mentions to a canonical set of real-world entities, additional links between entities and their contexts across document boundaries are established. Figure 1 illustrates this approach using snapshots of Wikipedia articles about actors and movies[1], where the articles are interconnected through jointly mentioned entities like "American Beauty".

From a syntactic point of view, the grammatical structures of the sentences are represented by hierarchically connected document substructures which become vertices in our BigText-KG. Specifically, *interlinked documents* form the basic entry points to our knowledge graph. These are decomposed into *sentences* which, in turn, consist of *clauses* (i.e., units of coherent information, each with an obligatory subject and a verbal or nominal predicate, and several optional (in)direct object(s), complement(s) and

[1] We use Wikipedia articles here to keep aligned with the experiments described in this paper. Note however that the choice of the document collection is not limited to any particular document type and can also combine heterogeneous natural-language resources, such as books, news, social networks, etc.

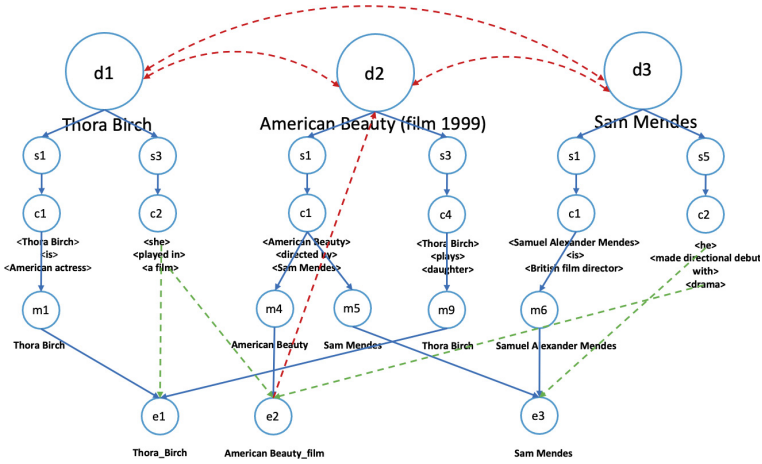

Fig. 2. Internal representation of the BigText graph of Fig. 1 as a property graph. The graph distinguishes five types of *vertices*: *documents* (d), *sentences* (s), *clauses* (c), *mentions* (m), and *entities* (e). The blue edges connect document's structural components. Red edges connect mentions to entities due to the named entity disambiguation and linking to real-life objects and, as a result, interlink the documents; green edges are "implicit" connections, linking mentions to entities due to syntactic dependencies and co-reference resolution.

adverbials that further contextualize the two mandatory components). Clauses further contain *mentions* of *named entities* (NEs) within their narrow clause contexts which then finally also capture the *relationships* among two or more such entities.

In natural language, entities are often referred to in various ways, requiring resolution and disambiguation before mapping them to a canonical set of real-world named entities (NEs) in a background KB like YAGO or Wikidata. This process establishes reliable links within and across documents' boundaries. Syntactic dependencies are crucial for disambiguation as they expand the fixed list of vocabulary variations to dynamically occurring local contexts. For instance, consider the sentence "*she* played Jane Burnham in the highly acclaimed *film American Beauty*." The apposition between *"film"* and *"American Beauty"* helps to link *"she"* directly to the real-world entity *"American Beauty"* instead of the generic concept *"film"*. Similarly, in the sentence "he made his directorial debut with the *drama American Beauty* (1999)," the apposition between *"drama"* and *"American Beauty"* enriches the entity with a semantic attribute describing its genre, which may be useful for sentiment analysis or further profiling of Sam Mendes' movie portfolio. Figure 2 shows a small subset of explicit and implicit relations that can be established between the entities Thora Birch, Sam Mendes and American Beauty using the structural representations of the respective documents from Fig. 1. Additionally, the analysis reveals different functions attributed to Thora Birch and Sam Mendes in American Beauty: Thora Birch *plays in* "American Beauty" while Sam Mendes *directs* "American Beauty" (an inverse relation obtained from a passive sentence "American Beauty is a 1999 American black comedy-drama film written by Alan Ball and *directed by* Sam Mendes.").

By incorporating more entities, exploring the syntactic and semantic dependencies between them and connecting the mentions to their real-world concepts, BigText incrementally builds a large-scale hybrid KG of highly interlinked and semantically enriched documents. This BigText-KG is designed to serve as a generic basis for a variety of text-analytical tasks such as searching and ranking, relation extraction, and question answering.

In this paper, we present a case study in which BigText is employed as an underlying knowledge graph of a *question answering* (QA) system, BigText-QA. When evaluated on questions involving multiple entities and relations between them, BigText-QA achieves competitive results with state-of-the-art QA systems like QUEST [32] and DrQA [8]. The rest of the paper is organized ass follows: Sect. 2 provides a survey of related work; Sect. 3 formally presents the BigText knowledge graph; Sect. 4 introduces the BigText question-answering system; Sect. 5 presents the experimental setup; Sect. 6 discusses the experimental results; and Sect. 7 finally concludes the paper.

2 Background and Related Work

In this section, we take a closer into the main QA approaches, which we broadly categorize based on their foundations: TextQA, KGQA (knowledge graph-based QA), and HybridQA (such as our BigText-QA), which seeks to integrate both textual and structured knowledge resources.

TextQA. TextQA approaches typically retrieve answers from raw unstructured text by extracting and aggregating information from relevant documents. Early systems like START [27] are representative of such approaches, while more recent ones like DrQA, DocumentQA [9], and R3 [48] employ neural-network techniques to enhance the matching capabilities. They leverage vast amounts of textual data and benefit from identifying semantic similarities but may struggle with complex queries calling for a concise structural representation.

KGQA. Traditionally, KG-based QA approaches transform a Natural Language (NL) input question into a logical representation by mapping NL phrases to various structured templates (e.g., in SPARQL). These templates can be executed against a query engine (e.g., by indexing RDF data) [4,12]. Recent KGQA techniques have improved upon this approach in two main ways: (1) incorporation of information retrieval (IR)-style relaxation on the templates to allow for selection and ranking of the relevant KG subgraphs in response to the input [7,14,20,51]; and (2) application of Neural Semantic Parsing (NSP) that converts the input question into a logical representation which, in turn, is translated into an actual query language understood by the KG [13,17,29,33,57]. While KG-based QA systems are strong in handling the logical structure of the question, the KG is inherently condensed and largely oblivious to the question's context which may weaken the system's performance.

HybridQA. Hybrid QA systems have been proposed for the sake of overcoming the limitations of the TextQA and KGQA paradigms while capitalizing on their respective strengths. A hybrid approach suggested in IBM seminal work [16] followed by [3],

obtains candidate answers from separate structured and unstructured resources. Among the systems that adopted such an integrated approach are [42], that combines external textual data and SPARQL-based templates for answering questions; [11] and [38,45] that employ neural networks to merge knowledge graphs (KGs) and textual resources into a common space using a universal-schema representation [41,46]. More recent research has explored the integration of KGs into large pre-trained language models (LLMs) and their application for open-domain question answering [26,38,52,53,56]. For example, UniK-QA [38] utilized the T5 model [39], a powerful LLM, to answer open-domain questions by leveraging heterogeneous sources.

Current state-of-the-art QA approaches achieve impressive results but also require vast amounts of data and significant time for training. Furthermore, they often need to dynamically integrate data retrieved from various external sources with KGs, adding to the systems' complexity. In contrast, BigText-QA circumvents the need for extensive training by building a unified property graph. This graph incorporates structured knowledge from disambiguated entities, while also preserving the original NL phrases that provide context and express relationships between the entities. This approach makes hybrid knowledge readily available for the QA process, thus reducing question-processing time. Furthermore, BigText-QA employs a Spark-based distributed architecture, enabling easy scalability and efficient handling of very large graphs.

3 BigText Knowledge Graph

Our BigText project is driven by the strong belief that natural-language text itself is the most comprehensive knowledge base we can possibly have; it just needs to be made machine-accessible for further processing and analytics.

Design and Implementation. BigText aims at processing large collections, consisting of millions of text documents. We currently employ Apache Spark [54] and its integrated distributed graph engine, GraphX [50], which allows us to model the entire collection as a unified property graph that can also be distributed across multiple compute nodes or be deployed on top of any of the common cloud architectures, if desired.

Property Graph. As depicted in Fig. 2, our BigText graph distinguishes five types of *vertices*: *documents* (d), *sentences* (s), *clauses* (c), *mentions* (m), and *entities* (e). Spark's GraphX allows us to associate an extensible list of properties for each vertex type, such that the (ordered) vertices are able to losslessly (and partly even redundantly) capture all the extracted information from a preconfigured Natural Language Processing (NLP) pipeline together with the original text sources. Figure 4 shows an internal representation of the property graph. For example, a document property stores the corresponding title and other relevant metadata, such as timestamp and source URL, while a mention vertex is augmented with morphological data, such as part-of-speech (POS) and lemma, the syntactic role within the sentence, as well as entity-type information, where applicable. Entity vertex property carries on the result of the mention disambiguation to a canonicalised entity.

Sentences, clauses and mentions form hierarchical substructures of documents, while links among different mentions (possibly from different clauses or sentences, or

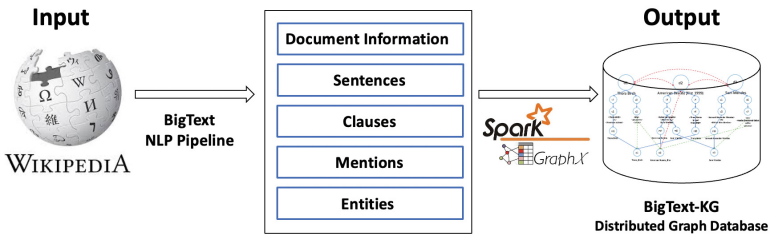

Fig. 3. High-level construction of the BigText knowledge graph.

```
class VertexProperty()

case class DocumentProperty(val
title: String, val timestamp,
val URL, val sentences:
Array[String]) extends
VertexPropert

case class SentenceProperty(val
content: String, val clauses:
Array[String]) extends
VertexProperty

case class ClauseProperty(val
content: String, val mentions:
Array[String]) extends
VertexProperty

case class
MentionProperty(val content:
String, val entities: String)
extends VertexProperty

case class EntityProperty(val
content: String) extends
VertexProperty

class EdgeProperty(val
source: String, val
destination: String)

var graph:
Graph[VertexProperty,
EdgeProperty](vertices,
edges)
```

Fig. 4. Case classes (in Scala) capturing the BigText-KG as a property graph in Spark's GraphX APIs.

even from different documents) to a same entity vertex in the background KB express additional coreferences. Recovered implicit relations resulting from appositions (e.g., "drama", and "film" with respect to "American Beauty") and co-reference resolution ("she" and "he" with respect to Thora Birch and Sam Mendes, respectively) are shown as green thin dashed lines in Fig. 2. Furthermore, the presence of *clause* vertices in combination with the disambiguated entity mentions, allows for dynamic extraction of the facts' subgraphs, containing mentions as vertices and the clauses' predicates as labeled edges.

NLP Pipeline. Before populating the property graph, documents in the collection are passed through a preconfigured and extensible NLP pipeline which decomposes the input into documents, sentences and clauses. Clauses are generated from sentences with an Open Information Extraction (OIE) technique. While each clause represents a semantically coherent block of entity mentions linked by a predicate, mentions first appear in their original lexical form without further linking to typed entities (such as `PER`, `ORG`, `LOC`) or unique knowledge base identifiers, for example WikiData IDs. Therefore, our pipeline also incorporates the steps of Named Entity Recognition (NER) and Named Entity Disambiguation (NED) as they are available from recent IE tools. Since clauses may have pronouns as their subject and/or object constituents, Corefer-

ence Resolution (CR) has been added to the NLP pipeline to increase the coverage of downstream analytical tasks. For example, linking *she* in "... she played Jane Burnham in the highly acclaimed film American Beauty" to *Thora Birch* establishes a connection between the two real-world entities, "American Beauty" and "Thora Birch", which can then further be explored.

Table 1. Annotators and background KBs used in the BigText NLP pipeline.

Annotation type	Tools
HTML parser	**Jsoup**[a]
Tokenization	Spacy
OpenIE	**ClausIE** [10], OpenIE5[b], OpenIE6 [28]
NER	**StanfordNLP**, Flair [43]
NED	**AIDA-Light** [37], REL [23], ELQ [31]
CR	SpanBERT:2018 [30], SpanBERT:2020 [24]
Background KB	**YAGO**, WikiData[c]

[a] https://jsoup.org/
[b] https://github.com/dair-iitd/OpenIE-standalone
[c] https://www.wikidata.org/

Projects which involve the stage of text (pre-)processing typically apply either an entire end-to-end suite of annotation tools, such as NLTK [5], StanfordNLP [35], SpaCy[2], or a specific component from it (which can also be substituted with a stand-alone or equivalent tool). Conversely, our text annotation pipeline does not limit the choice of annotators. We intend to use state-of-the-art target-specific components to minimise the risk of error propagation. This strategy allows us to adapt the selection of tools to the type of documents being processed (e.g., long documents corresponding to full-text Wikipedia articles versus short ones, such as Wikipedia articles' abstracts or news). Our implementation also allows for integrating outputs provided by different tools with the same annotation goal. In that way, the pipeline can be configured with further rules that prioritize either precision or recall (e.g., by considering either the intersection or the union of annotations). Table 1 depicts the annotation tools that have been integrated into the BigText NLP pipeline so far. Figure 3 depicts the entire construction process of the BigText knowledge graph (BigText-KG).

Table 2. BigText-KG statistics (in millions) for Wikipedia.

Documents	Sentences	Clauses	Mentions	Entities
5.3	97	190	283	2

Applications. In the following part of this paper, we focus on *question answering* (QA) as our main target application which relies on the BigText-KG as its underlying knowl-

[2] https://spacy.io/.

edge graph. We use full-text articles of an entire Wikipedia dump from 2019[3]. Statistics are shown in Table 2. Tools that have been used to process the version discussed here and used for the experiments are shown in Table 1 in bold font.

4 BigText Question Answering

The design of BigText-QA is based on QUEST, a graph-based question-answering system that specifically targets complex questions with multiple entities and relations. QUEST constructs a so-called *quasi-graph* by "googling" for relevant documents in response to an NL input question and by applying a proximity-based decomposition of sentences into $< \texttt{sub} >, < \texttt{pred} >, < \texttt{obj} >$ triplets (SPO). Similarly to our BigText-KG, leaf nodes of the quasi-graph are *mentions* (vertex labels in BigText), *relations* (edge properties in BigText) and *type* nodes (vertex properties in BigText). In QUEST, the latter are the result of a semantic expansion of mentions via the application of Hearst patterns [21] and/or lookups in an explicit mention-entity dictionary. In BigText, both the structural decomposition of the documents and their annotations have been provided by the preconfigured NLP pipeline.

Since our instance of the BigText-KG is built using Wikipedia as the text resource, our system consequently retrieves relevant Wikipedia documents by using Lucene as underlying search engine[4]. The top-10 of the retrieved documents then serve as pivots for the respective subgraph that is selected from the entire BigText-KG upon each incoming NL question. In summary, we translate such a BigText subgraph to a structure equivalent to QUEST's quasi-graph (depicted in Fig. 5) as follows:

Vertex Translation. Mention ("m") and entity ("e") vertices are directly translated from the BigText subgraph to the QUEST quasi-graph. These can be the subject and/or object of a clause. Type vertices ("t") are added based on the syntactic and semantic properties of the vertices, augmented with the application of Hearst patterns, following the QUEST approach. Predicate vertices ("p") are created out of the verbal component of a clause, while synonymous relation nodes are added using word/phrase embeddings (Sect. 4.1).

Edge Translation. Edges between predicates, mentions and disambiguated entities are directly translated from the BigText subgraph into the QUEST quasi-graph. Similarly to QUEST, we additionally introduce *type* and *alignment* edges from the respective vertex and edge properties in BigText. Type edges connect mention nodes with type nodes. For example, an organge edge between the mention node "Thora Birch" and type node "American actress" on the Fig. 5 is one such edge. It expresses the relation of type NP_1 *is* NP_2 captured by Hearst pattern. Alignment edges connect potentially synonymous mention nodes resolved to the same entity (thick blue edges between, for example, m_5 "Sam Mendes" and m_6 "Samuel Alexander Mendes", connected via e_3 to the entity "Sam_Mendes"), and potentially synonymous relation nodes such as "made directional debut" and "directed by"(dashed blues edges) in the same figure.

[3] https://dumps.wikimedia.org/enwiki/latest/.
[4] https://lucene.apache.org/core/.

4.1 Question-Answering Pipeline

In more detail, our QA pipeline processes an incoming NL question (or clue) as follows.

(1) The NL input question serves as a keyword query to Lucene which then retrieves the top-10 most relevant documents from the Wikipedia corpus. These documents are used as entry points to the BigText-KG to select a relevant subgraph that captures the questions' context; this subgraph includes all the documents' hierarchical substructures plus their links to the background KB.
(2) A syntactic parser (similar to QUEST) is applied to the question in order to identify its subject, predicate and object, which we refer to as the *question terms*.
(3) The subgraph is translated into QUEST's quasi-graph (see below for details).
(4) Vertices in the quasi-graph, which have high similarity with the question terms, become terminals (so-called "cornerstones" in QUEST). For example, "Thora Birch", "played in" and "plays" are examples of cornerstones, corresponding to question terms "Thora Birch" and "starred" (orange nodes in Fig. 5), respectively.
(5) Together with the quasi-graph and its weighted edges (see below), cornerstones constitute the input to QUEST's Group Steiner Tree (GST) algorithm [18] which is used to compute an answer set. A final ranking among the matching vertices in the answer set provides the ranked answers to the input question.

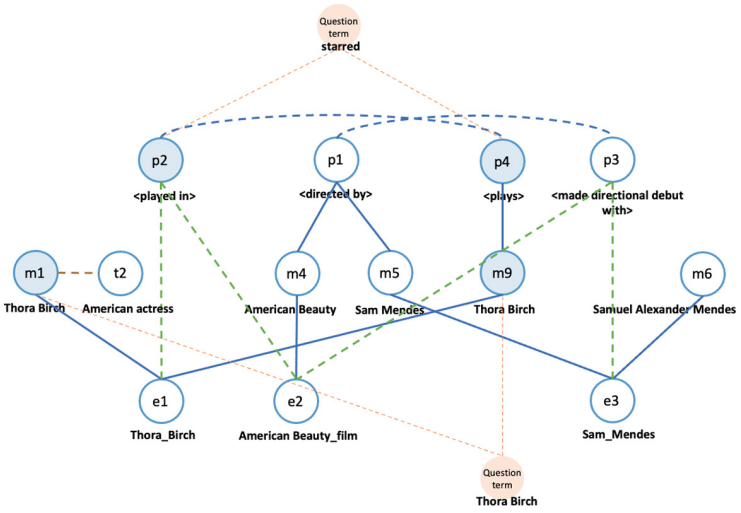

Fig. 5. QUEST quasi-graph for the question *"Which British stage director is best known for his feature-film directing debut, which starred Kevin Spacey, Annette Bening, and Thora Birch?"*. It results from the translation of the BigText (sub)graph in Fig. 2.

4.2 Weighting Schemes

Before we can proceed with the application of the GST algorithm and its answer-set calculation, both vertices and edges in QUEST's quasi-graph have to be assigned with weights, which (in our case) are derived from the relevant BigText subgraph.

- **Vertex weights** are defined by the similarity between the question terms and the vertices in the quasi-graph. Specifically, we adopt the two weighting schemes suggested in [32] and discuss their application.
- **Edge weights** are calculated depending on the edge type. The weight of an edge between a mention and a predicate vertex is the inverse of the distance between the two vertices in the BigText subgraph. Formally, this is defined as the number of words between a mention (i.e., subject and object) and a predicate of a clause vertex in the BigText subgraph. If two vertices are directly connected via multiple edges, the highest such weight is selected. Weights of alignment edges are calculated based on the semantic similarity between the vertices they connect (see Subsect. 4.3).

4.3 Similarities and Thresholds

Question terms are compared to the vertices in the quasi-graph according to their syntactic type as follows.

Jaccard Similarity. The AIDA dictionary [22] is a dictionary composed of a large number of entity-mention pairs in YAGO, where each mention is associated with the set of entities it may refer to; entities are represented by their unique identifiers. The "subject" and "object" question terms (see Sect. 4.1) are compared to the mention nodes using Jaccard similarity. Similarly, Jaccard similarity is between the entity sets associated with a mention vertex and the question term extracted from the input question. For mentions and question terms that could not be found in the AIDA dictionary, the Jaccard similarity is computed based on the plain string similarity between the two. In either case, the maximal value between a mention/entity and a question term selected as the vertex weight in the quasi-graph.

Cosine Similarity. Question terms identified as "predicates" are compared to the *predicate nodes* of the quasi-graph in a pair-wise manner using the Cosine similarity between their corresponding word embeddings[5]. For each predicate node in the quasi-graph, the maximal cosine similarity value of all pair-wise comparisons is selected as its weight.

Similarities for Alignment Edges. Once the node weights of the quasi-graph are computed, we can decide on the insertion of additional *alignment edges* into QUEST's quasi graph to further support the GST algorithm. An alignment edge is inserted if the similarity between two candidate vertices of the same type exceeds a pre-defined threshold. The similarity value then becomes the weight of the corresponding edge. Similarities between two mention vertices are again computed using Jaccard similarity (otherwise it is set to 1 if the two mentions are linked to the same entity), while the ones between predicate and type vertices are calculated using Cosine similarity.

[5] We use the default word2vec model [36] trained on Google news.

After the quasi-graph has been constructed, it is its largest connected component, together with the cornerstones, which is used as an input to the GST-algorithm.

Thresholds. All thresholds (calculated either by Jaccard or Cosine similarities) are set to 0.25 except the ones for the *predicate alignment edges* where we experiment with a range of values: 0.25, 0.375, 0.5, 0.6, 0.75 (see Table 3). We remark that our threshold policy is different from the one applied by QUEST, where all thresholds are the same and set to 0.5.

After the quasi-graph construction, we proceed with the answer-set computation, ranking and filtering. These steps are performed exactly as in the QUEST framework.

5 Experiments

All experiments reported in this paper are conducted on a single large Intel Xeon Platinum server with 2.4 GHz, 192 virtual cores and 1.2 TB of RAM, holding the entire BigText-KG in main memory. All translation steps are performed using PySpark 2.4.1 [55] for transforming Spark's GraphX RDDs into the relevant BigText subgraph via parallel processing. The BigText subgraph is translated into QUEST's quasi-graph by a second Python library, NetworkX 2.8 [19].

We use two benchmark datasets for the evaluation: CQ-W [1] and TriviaQA [25]. Regarding CQ-W, we remove questions whose answers are not present in the Wikipedia-based BigText subgraph, which is the case for about 25% of the questions[6]. The remaining 75% of the questions are used for the comparative evaluation. As for TriviaQA, we randomly select 79 questions from the development set (`wikipedia-dev.json`). CQ-W is a curated dataset of question-and-answer pairs, which consists of 150 complex questions from Wiki-Answers [15]. TriviaQA is a large-scale dataset made of complex and compositional questions and corresponding gold answers.

To run the experiments, we feed the top-10 documents selected from Wikipedia by Lucene both to the original QUEST engine and to BigText-QA in order to ensure a fair comparison. We also quote the results achieved by DrQA on the CQ-W dataset as a further baseline. As opposed to QUEST and BigText-QA, DrQA is a neural-network-based QA system, and thus represents another class of QA systems. DrQA is trained on the SQuAD [40] question-and-answer set which is also based on a subset of Wikipedia articles.

6 Result and Discussion

The main scoring metric for evaluating the QA systems is Mean Reciprocal Rank (MRR), calculated as $MRR = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{rank_i}$, where Q is the number of questions and $rank_i$ is the rank of the first correct answer for the i-th question. Other important metrics include Precision@1 (P@1) and Hit@5. P@1 represents the precision of the

[6] This decision was motivated by the fact that, for those questions, none of the top-10 documents returned by Lucene actually contained the answer.

Table 3. Comparison between BigText-QA, QUEST and DrQA on the CQ-W and TriviaQA datasets.

Dataset	System	Cosine	#Vertices	#Edges (10^5)	MRR	P@1	Hit@5
CQ-W	BigText-QA	0.250	1,276	7.234	0.387	0.324	0.441
		0.375	1,276	7.234	0.387	0.324	0.441
		0.500	**1,268**	6.727	**0.398**	**0.342**	**0.423**
		0.600	**579**	0.510	**0.264**	**0.198**	**0.297**
		0.750	210	0.030	0.140	0.081	0.189
	QUEST	0.500	2,385	13.580	0.464	0.423	0.495
		0.600	**1,267**	0.609	0.329	0.279	0.369
		0.750	**642**	0.032	0.181	0.099	0.279
	DrQA	-	-	-	0.120	0.171	0.315
Trivia-QA	BigText-QA	0.375	840	2.073	0.412	0.342	0.494
		0.500	**838**	1.968	**0.412**	**0.342**	**0.468**
		0.600	**365**	0.163	**0.258**	**0.190**	**0.316**
		0.750	121	0.007	0.130	0.063	0.190
	QUEST	0.500	1,710	4.025	0.425	0.380	0.468
		0.600	**968**	0.241	0.285	0.215	0.329
		0.750	**490**	0.025	0.198	0.139	0.241

top-ranked document retrieved, while Hit@5 is 1 if one of the top 5 results includes the correct answer. These metrics offer valuable insights into the system's performance. A higher score indicates better performance for the QA system. The results are shown in Table 3 and Table 4. In these tables, *Cosine* refers to the edge threshold which is used to select *predicate alignment edges*; *#vertices* and *#edges* refer to the largest connected component of the quasi-graph which is used as input to the GST-algorithm. *MRR*, *P@1*, *Hit@5* (as well as the number of nodes and edges in the respective quasi-graphs) are averaged across all questions.

Table 3 shows that BigText-QA achieves very competitive results, usually generating more compact yet denser graphs compared to QUEST. This compactness positively impacts the results, as it involves fewer answer candidates. QUEST slightly outperforms BigText-QA when its quasi-graph has nearly twice as many nodes. However, in cases where both systems generate quasi-graphs of comparable order (BigText-QA with $Cosine = 0.5$, QUEST with $Cosine = 0.6$, and BigText-QA with $Cosine = 0.6$, QUEST with $Cosine = 0.75$, shown in bold font in Table 3), BigText-QA outperforms QUEST on both question sets, CQ-W, and TriviaQA.

Table 3 indicates two ways of achieving close results using the GST algorithm: either by having sufficient, even "poorly" connected vertices (QUEST), or by having fewer but better connected vertices (BigText-QA). Different quasi-graph configurations in both systems arise from distinct underlying NLP pre-processing of input documents, particularly in the decomposition of sentences into clauses. BigText-QA yields fewer but more accurate vertices, facilitating the generation of dense graphs even with the increasing threshold values for edge insertion, which positively affects the performance.

Note, that the effect of changing the edge threshold below 0.5 is negligible (first two rows in the Table 3). However, raising it from 0.5 to 0.6 significantly decreases the number of edges. This may be due to the word2vec model itself, as a low similarity

Table 4. Comparison of QUEST and BigText-QA over different categories of questions in CQ-W.

Type	System	Cosine	#Vertices	#Edges(10^5)	MRR	P@1	Hit@5
People	BigText-QA	**0.500**	**1,375**	9.451	**0.388**	**0.333**	**0.407**
		0.600	**654**	0.667	**0.273**	**0.185**	**0.333**
		0.750	250	0.035	0.173	0.111	0.204
	QUEST	0.500	2,600	19.396	0.448	0.389	0.500
		0.600	**1,384**	0.793	0.304	0.259	0.333
		0.750	**741**	0.037	0.159	0.074	0.259
Movie	BigText-QA	**0.500**	**1,428**	4.843	0.353	0.333	0.333
		0.600	**657**	0.507	0.242	0.200	0.233
		0.750	185	0.029	0.074	0.033	0.100
	QUEST	0.500	2,636	9.279	0.504	0.500	0.500
		0.600	**1,381**	0.496	**0.441**	**0.433**	**0.433**
		0.750	**589**	0.027	**0.094**	**0.067**	**0.067**
Place	BigText-QA	**0.500**	**756**	2.059	**0.580**	**0.444**	**0.722**
		0.600	254	0.107	0.293	0.167	0.389
		0.750	129	0.006	0.108	0.056	0.222
	QUEST	0.500	1,393	3.126	0.498	0.444	0.556
		0.600	**729**	0.168	0.315	0.167	0.500
		0.750	414	0.013	0.359	0.222	0.500

threshold results in numerous "weak" alignment edges. A change from 0.5 to 0.6 is typically where the model becomes more discriminative, leading to a smaller number of synonymous edges. This trend continues with a further increase in the threshold to 0.75.

When compared to DrQA, both QUEST and BigText-QA outperform it, as they are designed to handle complex questions and incorporate evidence from multiple documents. In contrast, DrQA embodies an IR-based approach to QA, expecting the answer to a question to be narrowed down to a specific text span within a single document that closely matches the question.

To gain a more detailed understanding of the performance of BigText-QA versus QUEST, we divided the CQ-W set into six question categories: *People*, *Movie*, *Place*, *Others*, *Language*, *Music*. However, the last three categories turned out too small to be representative (containing only 4, 2 and 3 questions, respectively). We therefore focus on the first three categories (*People*, *Movie*, *Place*) in Table 4. Here again we highlight in boldface the lines which show the results obtained by both the systems on the quasi-graphs of comparable order. Consistent with the overall results, BigText-QA and QUEST demonstrate similar performance patterns in *People*- and *Place*-related questions as we discussed previously. However, BigText-QA does not compare favorably in *Movie*-related questions. We leave an in-depth investigation of this result as a future work.

7 Conclusion

In this paper, we have presented BigText-QA—a question answering system that uses a large-scale hybrid knowledge graph as its knowledge base. To this end, BigText-QA outperforms DrQA, a state-of-the-art neural-network-based QA system, and achieves competitive results with QUEST, a graph-based unsupervised QA system that inspired the design of BigText-QA. We see these results as a proof-of-concept for our hybrid knowledge representation which captures both textual and structured components in a unified manner in our BigText-KG approach.

Acknowledgments. This work was funded by FNR (Grant ID: 15748747). We thank Rishiraj Saha Roy and his group at the Max Planck Institute for Informatics for their helpful discussions and their support on integrating QUEST with our BigText graph.

References

1. Abujabal, A., Yahya, M., Riedewald, M., Weikum, G.: Automated template generation for question answering over knowledge graphs. In: WWW (2017)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: ISWC (2007)
3. Baudiš, P., Šedivỳ, J.: Modeling of the question answering task in the yodaqa system. In: CLEF (2015)
4. Berant, J., Chou, A., Frostig, R., Liang, P.: Semantic parsing on freebase from question-answer pairs. In: EMNLP (2013)
5. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc. (2009)
6. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: a collaboratively created graph database for structuring human knowledge. In: SIGMOD (2008)
7. Bordes, A., Chopra, S., Weston, J.: Question answering with subgraph embeddings. arXiv preprint arXiv:1406.3676 (2014)
8. Chen, D., Fisch, A., Weston, J., Bordes, A.: Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051 (2017)
9. Clark, C., Gardner, M.: Simple and effective multi-paragraph reading comprehension. arXiv preprint arXiv:1710.10723 (2017)
10. Corro, L.D., Gemulla, R.: Clausie: clause-based open information extraction. In: WWW (2013)
11. Das, R., Zaheer, M., Reddy, S., McCallum, A.: Question answering on knowledge bases and text using universal schema and memory networks. arXiv preprint arXiv:1704.08384 (2017)
12. Diefenbach, D., Lopez, V., Singh, K., Maret, P.: Core techniques of question answering systems over knowledge bases: a survey. KAIS (2018)
13. Dong, L., Lapata, M.: Language to logical form with neural attention. arXiv preprint arXiv:1601.01280 (2016)
14. Dong, L., Wei, F., Zhou, M., Xu, K.: Question answering over freebase with multi-column convolutional neural networks. In: ACL-IJCNLP (2015)
15. Fader, A., Zettlemoyer, L., Etzioni, O.: Paraphrase-driven learning for open question answering. In: ACL (2013)
16. Ferrucci, D., et al.: Building watson: an overview of the deepqa project. AI Mag. (2010)
17. Fu, B., Qiu, Y., Tang, C., Li, Y., Yu, H., Sun, J.: A survey on complex question answering over knowledge base: recent advances and challenges. arXiv preprint arXiv:2007.13069 (2020)

18. Garg, N., Konjevod, G., Ravi, R.: A polylogarithmic approximation algorithm for the group steiner tree problem. J. Algorithms (2000)
19. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab. (LANL) (2008)
20. Hao, Y., et al.: An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In: ACL (2017)
21. Hearst, M.A.: Automatic acquisition of hyponyms from large text corpora. In: COLING (1992)
22. Hoffart, J., et al.: Robust disambiguation of named entities in text. In: EMNLP (2011)
23. van Hulst, J.M., Hasibi, F., Dercksen, K., Balog, K., de Vries, A.P.: REL: an entity linker standing on the shoulders of giants. In: SIGIR (2020)
24. Joshi, M., Chen, D., Liu, Y., Weld, D.S., Zettlemoyer, L., Levy, O.: Spanbert: improving pre-training by representing and predicting spans. Trans. Assoc. Comput. Linguistics (2020)
25. Joshi, M., Choi, E., Weld, D.S., Zettlemoyer, L.: Triviaqa: a large scale distantly supervised challenge dataset for reading comprehension. arXiv preprint arXiv:1705.03551 (2017)
26. Ju, M., Yu, W., Zhao, T., Zhang, C., Ye, Y.: Grape: knowledge graph enhanced passage reader for open-domain question answering. arXiv preprint arXiv:2210.02933 (2022)
27. Katz, B., Felshin, S., Lin, J.J., Marton, G.: Viewing the web as a virtual database for question answering. In: New Directions in Question Answering (2004)
28. Kolluru, K., Adlakha, V., Aggarwal, S., Mausam, Chakrabarti, S.: OpenIE6: iterative grid labeling and coordination analysis for open information extraction. In: EMNLP (2020)
29. Lan, Y., Jiang, J.: Query graph generation for answering multi-hop complex questions from knowledge bases. In: ACL (2020)
30. Lee, K., He, L., Zettlemoyer, L.: Higher-order coreference resolution with coarse-to-fine inference. CoRR (2018)
31. Li, B.Z., Min, S., Iyer, S., Mehdad, Y., Yih, W.: Efficient one-pass end-to-end entity linking for questions. In: EMNLP (2020)
32. Lu, X., Pramanik, S., Saha Roy, R., Abujabal, A., Wang, Y., Weikum, G.: Answering complex questions by joining multi-document evidence with quasi knowledge graphs. In: SIGIR (2019)
33. Luo, K., Lin, F., Luo, X., Zhu, K.: Knowledge base question answering via encoding of complex query graphs. In: EMNLP (2018)
34. Manning, C.D.: An introduction to information retrieval. Cambridge University Press (2009)
35. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The stanford corenlp natural language processing toolkit. In: ACL (2014)
36. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS (2013)
37. Nguyen, D.B., Hoffart, J., Theobald, M., Weikum, G.: AIDA-light: high-throughput named-entity disambiguation. In: LDOW (2014)
38. Oguz, B., et al.: Unik-qa: unified representations of structured and unstructured knowledge for open-domain question answering. arXiv preprint arXiv:2012.14610 (2020)
39. Raffel, C., et al.: Exploring the limits of transfer learning with a unified text-to-text transformer. JMLR (2020)
40. Rajpurkar, P., Zhang, J., Lopyrev, K., Liang, P.: Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250 (2016)
41. Riedel, S., Yao, L., McCallum, A., Marlin, B.M.: Relation extraction with matrix factorization and universal schemas. In: NAACL-HLT (2013)
42. Savenkov, D., Agichtein, E.: When a knowledge base is not enough: question answering over knowledge bases with external text data. In: SIGIR (2016)
43. Schweter, S., Akbik, A.: FLERT: document-level features for named entity recognition. CoRR (2020)

44. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: a core of semantic knowledge. In: WWW (2007)
45. Sun, H., Bedrax-Weiss, T., Cohen, W.W.: Pullnet: open domain question answering with iterative retrieval on knowledge bases and text. arXiv preprint arXiv:1904.09537 (2019)
46. Verga, P., Belanger, D., Strubell, E., Roth, B., McCallum, A.: Multilingual relation extraction using compositional universal schema. arXiv preprint arXiv:1511.06396 (2015)
47. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. CACM (2014)
48. Wang, S., et al.: R 3: Reinforced ranker-reader for open-domain question answering. In: AAAI (2018)
49. Weikum, G., Dong, X.L., Razniewski, S., Suchanek, F., et al.: Machine knowledge: Creation and curation of comprehensive knowledge bases. Found, Trends Databases (2021)
50. Xin, R.S., Crankshaw, D., Dave, A., Gonzalez, J.E., Franklin, M.J., Stoica, I.: GraphX: Unifying data-parallel and graph-parallel analytics. CoRR (2014)
51. Yao, X., Van Durme, B.: Information extraction over structured data: Question answering with freebase. In: ACL (2014)
52. Yasunaga, M., Ren, H., Bosselut, A., Liang, P., Leskovec, J.: Qa-gnn: reasoning with language models and knowledge graphs for question answering. arXiv preprint arXiv:2104.06378 (2021)
53. Yu, D., et al.: Kg-fid: infusing knowledge graph in fusion-in-decoder for open-domain question answering. arXiv preprint arXiv:2110.04330 (2021)
54. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: HotCloud (2010)
55. Zaharia, M., et al.: Apache Spark: a unified engine for big data processing. CACM (2016)
56. Zhang, L., et al.: A survey on complex factual question answering. AI Open (2023)
57. Zhu, S., Cheng, X., Su, S.: Knowledge-based question answering by tree-to-sequence learning. Neurocomputing (2020)