Knowledge Questions from Knowledge Graphs

Dominic Seyler
University of Illinois at Urbana-Champaign
dseyler2@illinois.edu

Mohamed Yahya
Bloomberg LP, London
myahya6@bloomberg.net

Klaus Berberich
Max Planck Institute for Informatics
HTW Saar
kberberi@mpi-inf.mpg.de

ABSTRACT

We address the problem of automatically generating quiz-style knowledge questions from a knowledge graph such as DBpedia. Questions of this kind have ample applications, for instance, to educate users about or to evaluate their knowledge in a specific domain. To solve the problem, we propose a novel end-to-end approach. The approach first selects a named entity from the knowledge graph as an answer. It then generates a structured triple-pattern query, which yields the answer as its sole result. If a multiple-choice question is desired, the approach selects alternative answer options as distractors. Finally, our approach uses a template-based method to verbalize the structured query and yield a natural language question. A key challenge is estimating how difficult the generated question is to human users. To do this, we make use of historical data from the Jeopardy! quiz show and a semantically annotated Web-scale document collection, engineer suitable features, and train a logistic regression classifier to predict question difficulty. Experiments demonstrate the viability of our overall approach.

ACM Reference format:
Dominic Seyler, Mohamed Yahya, and Klaus Berberich. 2017. Knowledge Questions from Knowledge Graphs. In *Proceedings of ICTIR '17, Amsterdam, Netherlands, October 01-04, 2017,* 8 pages.
DOI: .1145/3121050.3121073

1 INTRODUCTION

In this work, we address the problem of generating quiz-style knowledge questions from knowledge graphs (KGs). As shown in Figure 1, starting from a KG and a topic such as *US Presidents*, we generate a quiz question whose unique answer is an entity from that topic. The question starts its life as an automatically generated triple-pattern query, which our system verbalizes. Each generated question is adorned with a difficulty level, providing an estimate for how hard it is to answer, and optionally a set of distractors, which can be listed alongside the correct answer to obtain a multiple-choice question. Our system is able to judge the impact of distractors on the difficulty of the resulting multiple-choice question.

Applications of automatically generated knowledge questions include education and evaluation. One way to educate users about a specific domain (e.g., Politics) is to prompt them with questions, so that they pick up facts as they try to answer – reminiscent of flash cards used by pupils. When qualification for a task needs to be ensured, such as knowledge about a specific domain, automatically generated knowledge questions can serve as a qualification test. Crowdsourcing is one concrete use case as outlined in [35]. Likewise, knowledge questions can serve as a form of CAPTCHA to exclude likely bots.

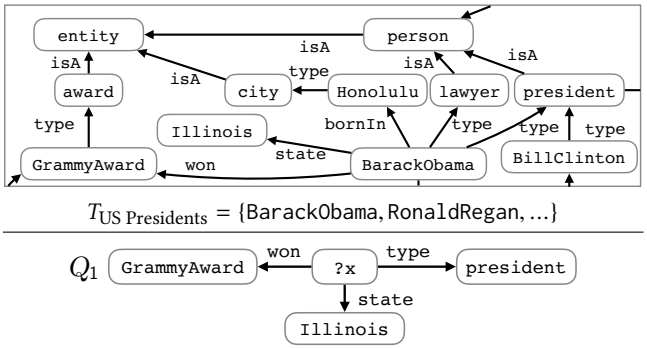

Figure 1: A KG fragment, a topic, a hard question, and two distractors (one easy and one hard).

Challenges. To discriminate how much people know about a domain it is typical to ask progressively more difficult questions. In our setting, this means that we need to automatically quantify the difficulty of a question. This is not trivial as it requires us to consider multiple signals and their interaction. One might consider all questions whose answer is `BarackObama` to be easy, as he is a prominent entity. However, few people would know that he won a `GrammyAward`. Therefore, signals that predict question difficulty need to be identified and combined in a meaningful manner.

An answer should be easy to verify automatically and disputes about the correctness of an answer should be minimal. We envision a setting with minimal human involvement, which we achieve by ensuring that each question has exactly one correct answer. We deal with possible variation in user input (e.g., *'Barack Obama'* vs *'Barack H. Obama'*) by turning fill-in-the-blank questions into multiple-choice questions. Here, we carefully consider the impact distractors have on question difficulty.

A final challenge is the production of well-formed questions that look natural. Such questions provide a better experience to users and make them hard to identify as having been automatically generated. Two important factors here are question coherence and linguistic variety. For example, while a *KG* may classify `BarackObama` as an `entity` and a `formerSenator`, we use the latter in asking

ICTIR '17, Amsterdam, Netherlands

DOI: .1145/3121050.3121073

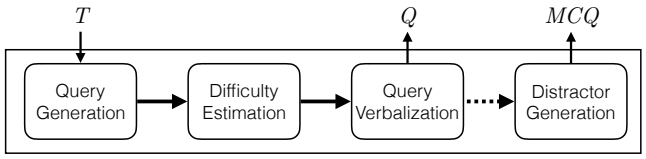

Figure 2: Question generation pipeline.

about him, as the first is unnatural. Similarly, while a relation of an actor to a movie is called `actedIn`, we want to vary it's expression (e.g., *'acted in'* or *'starred in'*).

Contributions. We propose a novel end-to-end approach for generating quiz-style knowledge questions from knowledge graphs. Our approach has three major components: query generation, difficulty estimation, and query verbalization. In a setting where multiple-choice questions are desired, a fourth component can generate distractors and quantify their impact on question difficulty. Figure 2 depicts our pipeline.

The query generation component generates a structured query that will serve as the basis of the final question shown to a human. By starting from a structured query, we are able to generate questions that are certain to have one unique, correct answer in our KG. We discuss challenges that need to be addressed so that the resulting cues are meaningful.

To estimate the difficulty of a question we leverage different signals about named entities, which we derive from a Web-scale document collection annotated with named entities from the KG. We use these signals as features to train a difficulty classifier with supervision obtained from more than thirty years of data from the Jeopardy! quiz show.

Since our questions start their life as structured queries over the KG, we verbalize them to generate a corresponding natural language question. Following earlier work, we adopt a template-based approach. However, we extend this approach with automatically mined paraphrases for relations and classes in the KG, ensuring diversity in the resulting natural language questions.

Outline. The rest of this paper unfolds as follows. Section 2 introduces preliminaries and provides a formal problem statement. Section 3 describes how a SPARQL query can be generated that has a unique answer in the KG. Our approach for estimating the difficulty of the generated query is presented in Section 4. Section 5 describes how the query can be verbalized into natural language. Extensions for multiple-choice questions are described in Section 6. Section 7 lays out the setup and results of our experiments. We put our work in the context of prior research in Section 8, before concluding in Section 9.

2 PRELIMINARIES & PROBLEM STATEMENT

Knowledge Graphs (KGs) such as as Freebase [9] and Yago [38] describe *entities* E (e.g., `BarackObama`) by connecting them to other entities, *types* T — also called classes (e.g., `president`, `leader`), and *literals* L (e.g., *'1985-02-05'*) using *predicates* P (e.g., `bornIn`, `birthdate`, `type`). A KG is thus a set of facts (or triples), $\{f \mid f \in E \cup T \times P \times E \cup T \cup L\}$. A triple can also be seen as an instance of a binary predicate, with the first argument called *subject* and the second called *object*, hence the model is referred to as subject-predicate-object (SPO) model. Figure 1 shows a KG fragment.

Pattern matching is used to query a KG. Given a set of variables V that are always prefixed with a question mark (e.g., ?x), a *triple-pattern-query* is a set of triple patterns $Q = \{q \mid q \in V \cup E \cup T \times V \cup P \times V \cup E \cup T \cup L\}$. An answer a to a query is a total mapping of variables to items in the KG such that the application of a to each q results in a fact in the KG. In our setting, inspired by Jeopardy!, we restrict ourselves to queries having a single variable for which a unique answer exists in the KG.

We use Yago2s [39] as our reference KG, which contains 2.6M entities, 300K types organized into a type hierarchy, and more than 100 predicates that form over 48M facts. Yago entities are associated with Wikipedia entries, whereas Yago types correspond to WordNet synsets [13] or Wikipedia categories. For estimating question difficulty, we utilize the ClueWeb09/12 document collections and the FACC annotations provided by Google [16]. The latter provide semantic annotations of disambiguated named entities from Freebase, which we map to Yago2s via their Wikipedia article.

Jeopardy! is a popular U.S. TV quiz show that features questions referred to as clues. Each clue comes with a monetary value, corresponding to the amount added to a contestant's balance when answering correctly. We reckon that monetary values correlate with human performance and thus question difficulty – a hypothesis which we investigate in Section 4.

Problem Statement. Put formally, our objective is to automatically generate a question Q whose unique answer is an entity $e \in \mathfrak{T}$ which can be supported by facts in the KG. $\mathfrak{T}$ is a thematic set of entities called a *topic*, which allows us to control the domain from which knowledge questions are generated (e.g., *American Politics*). Moreover, we assume a predefined set of *difficulty levels* $D = \{d_1, ..., d_n\}$ with a strict total order $<$ defined over its elements, and we want to estimate the difficulty of providing the answer a to Q, denoted $diff(Q, a)$. An extension of the above problem which we also deal with in this work is the generation of *multiple choice questions* (MCQs), where the task is to extend a question Q into an *MCQ* by generating a set of incorrect answers, called *distractors*, and quantifying their difficulty.

In our concrete instantiation of the above problem, we use Wikipedia categories as topics and Yago2s as our KG. As a first attempt to address the above problem, we consider a setting with two difficulty levels, $D = \{easy, hard\}$, where $easy < hard$. For our purposes, a *question* is any natural language sentence that requires an answer. It can be an interrogative sentence, or a declarative one in the style of Jeopardy! clues.

Generality. All methods and tools proposed in this work are general enough to apply to a setting other than ours of Jeopardy! and Yago. Our approach can be applied to any KG that is represented as triples and abides by the standards described earlier in this section. In addition, lexical knowledge is required in the form of surface forms for entities (e.g., `BarackObama` → "Barack Obama") and relations (e.g., `actedIn` → "starred in"). For difficulty estimation, our approach requires a question-answer corpus with annotated difficulties. Statistics are required about salience of entities and coherence of entity pairs, which can be estimated using external corpora like Wikipedia.

3 QUERY GENERATION

The first stage is the generation of a query that has a unique answer in the KG. This query serves as the basis for generating a question that will be shown to human contestants. The unique answer needs to be provided by the user in order to correctly answer the question. Ensuring that a question has a single answer simplifies verification.

The input to the query generation step is a topic $\mathfrak{T}$ (e.g., US Presidents). The unique answer to the generated query will be an entity $e \in \mathfrak{T}$ randomly drawn from the KG (e.g., `BarackObama`). After drawing e, a subset of triple patterns is selected from the KG where e is either subject or object in the pattern. These triples form the question's content.

The selection of triple patterns is guided by the following desiderata: i) the query should contain at least one type triple pattern, which is crucial when verbalizing the query to generate a question, and ii) entities mentioned in the query should not give any obvious clues about the answer entity. In what follows we present the challenges in achieving each of these desiderata, and our solutions to these challenges.

3.1 Answer Type Selection

Questions asking for entities always require a type that is either specified implicitly (e.g., *'who'* for `person`) or explicitly (e.g., *"Which president ..."*). KGs tend to contain a large number of types and typically associate an entity with multiple types. Some of these types typically appear in text talking about an entity (e.g., `president`, `lawyer`). Other types, however, are artifacts of attempts to have an ontologically complete and formally sound type system. Such types are meaningful only in the context of a type system, but not on their own (e.g., the type `entity`).

To address the problem of selecting a type for an answer entity, we use our entity-annotated corpus to capture the salience of a semantic type t for an entity e, denoted $s(t, e)$. We start by collecting occurrences of an entity e along with *textual types* to which it belongs t_{text} in our entity-annotated corpus. We use the following patterns, inspired by Hearst [19], to collect (t_{text}, e) pairs:

P1: `ENTITY` (*'is a'*|*'is an'*|*', a'*|*'and other'*|*'or other'*) TYPE
"`BarackObama` *and other presidents attended the ceremony.*"
P2: TYPE (*'like'*|*'such as'*|*'including'*|*'especially'*|) `ENTITY`
"*...several attorneys including* `BarackObama`"

The next step is to disambiguate (t_{text}, e) pairs to (t, e) pairs — note that entities are already disambiguated in the corpus, so we only need to disambiguate t_{text} to a semantic type t in the KG. Relying on the fact that our semantic types are WordNet synsets, we use the WordNet lexicon (e.g., {*lawyer, attorney*} → `lawyer`) for generating a set of semantic type candidates for a given textual type. We then use a heuristic where a textual type t_{text} paired with an entity e is disambiguated to a semantic type t if i) t is in the set of candidates for t_{text} and ii) $e \in t$. We compute salience $s(t, e)$ as the relative frequency with which the disambiguated (t, e) pair was observed in our corpus. To select a type for the answer entity e, we draw one of its types randomly based on $s(t, e)$.

3.2 Triple Pattern Generation

We now have an answer entity e and one of its semantic types t that will be used to refer to e in the question. We now need to create a query whose unique answer over the KG is e. Creating a query means selecting facts where e is either the subject or object and turning these into triple patterns by replacing e with a variable (`?x`). Not all facts can be used here, as some reveal too much about the answer and render the question too trivial. Other facts will be redundant given the facts already used.

Elimination of Textual Overlap with the Answer. The first restriction we impose on a fact is that the surface forms of entities that appear in it cannot have any textual overlap with surface forms of the answer entity. The question *"Which president is married to Michelle Obama?"* reveals too much about the answer entity. For overlap, we look at the set of words in the surface forms, excluding common stop words.

Elimination of Redundant Facts. Given a set of facts that has been chosen, a new fact does not always add new information. Keeping this redundant fact in a query will allow humans to clearly identify a question as being automatically generated. To eliminate this issue, we check each new type fact against all existing ones. If the new type is a supertype (e.g., `person`) of an existing one (e.g., `president`), we discard it.

4 DIFFICULTY ESTIMATION

We now describe our approach to estimating the difficulty of answering the knowledge query generated in Section 3. There are several, seemingly contradictory, signals that affect the difficulty of a question. As discussed earlier, one might expect any question asking for a popular entity such as `BarackObama` to be an easy one. However, if we were to ask *"Which president from Illinois won a Grammy Award?"*, few people are likely to think of `BarackObama`. We use a classification model trained on a corpus of questions paired with their difficulties to predict question difficulty. Note that the difficulty is computed based on the query and not its verbalization. Our goal here is to create questions that measure factual knowledge rather than linguistic ability.

Since we rely on supervised training for difficulty estimation, we make the natural assumption that difficulty labels in the 'training' and 'testing' questions are drawn from the same underlying distribution for some target audience. We also assume that for this population, it is possible to capture the difficulty of a question. As evidence for this, in the Jeopardy! dataset [1] we find a positive correlation between the attempted questions for a certain difficulty-level and the number of times a question of this difficulty-level could not be answered. For the five difficulty-levels ($200, $400, $600, $800, $1000), 4.46%, 8.35%, 12.69%, 17.82% and 25.69% of the questions could not be answered, respectively.

4.1 Preparing Training Data

We use the Jeopardy! quiz-game show data described in Section 2 for training and testing our difficulty-estimation classifier. The larger goal is to estimate the difficulty of answering queries generated from a knowledge graph, so we restrict ourselves to a subset of the Jeopardy! questions answerable from Yago [38], which we collected as described below.

We say a question is answerable from Yago if i) all entities mentioned in the question and its answer are in Yago, and ii) all relations connecting these entities are captured by Yago. To find these questions, we automatically annotate Jeopardy! questions with Yago entities using the AIDA tool for named entity disambiguation [21]. An example of a disambiguated question is:

ShahJahan *built this complex in* Agra *to immortalize* MumtazMahal, *his favorite wife.* TajMahal

We retain an entity-annotated question if i) its answer can be mapped to a Yago entity, ii) its body has at least one entity (the one that will be given in the question, not the answer), and iii) considering all entities in the question and the answer entity, each entity can be paired with another entity to which it has a direct relation in Yago. The last condition ensures that we have questions that can be captured by the relationships in Yago. However, it does not identify this relation, and such a match may be spurious. Since this is hard to establish automatically, we invoke humans at this point.

We run a crowdsourcing task on the questions that survive the above automated annotation and filtering procedure. The task is to label a question/answer pair as *Good* if i) all entities in the question have been captured and disambiguated correctly, ii) the question can be captured by relations in Yago, and iii) the answer is a unique one. The crowdsourcing task ran until we obtained a total of 500 questions that we use in our experiments.

4.2 Difficulty Classifier

After obtaining the training/testing data, we turn our attention to building the difficulty classifier and its features. Formally, our goal is to learn a function $diff(Q, e) \in \{easy, hard\}$ that estimates the difficulty of providing the answer e to the query Q. We use logistic regression due to the ease with which it can be trained and because it allows easy inspection of feature weights, which proved helpful during development. As we are dealing with a binary classification case ($easy, hard$), we train our model to learn the probability of the question being an *easy* one, $P(diff(Q, e) = easy)$, and set a decision boundary at 0.5. We judge a question to be *easy* if $P(diff(Q, e) = easy) > 0.5$ and *hard* otherwise.

The model, however, only works if provided with the right features. Table 1 provides a summary of our features. The key ingredients in our feature repertoire are:

Entity Salience (ϕ) is a normalized score that is used as a proxy for an entity's popularity. As our entities come from Wikipedia, we use the Wikipedia link structure to compute entity salience as the relative frequency with which the Wikipedia entry for an entity is linked to from all other entries. We also consider salience on a per-coarse-semantic-type basis. The second group of Table 1 defines a set of templates. We consider the coarse semantic types `person`, `location`, and `organization` and define a fourth coarse semantic type `other` that collects entities not in any of the three aforementioned coarse types (e.g., movies, inventions). Having specialized features for individual coarse-grained types allows us to take into account some particularities of these coarse types. For example, locations tend to have disproportionately high salience. By having a feature that accounts for this specific semantic type, we can mitigate this. Without this feature, a location in a question would always result in our classifier labeling the question as easy.

Table 1: Difficulty estimator features and their description. $\mathcal{T}$ is one of `person`, `organization`, `location`, or `other`.

Feature	Description
Entity Salience	
ϕ_{target}	answer entity salience
ϕ_{min}	min. salience of question entities
ϕ_{max}	max. salience of question entities
ϕ_{Σ}	sum over salience of entities
ϕ_{μ}	mean salience of question and answer entities
ϕ_{μ}^{q}	mean salience of entities in question
Per-coarse-semantic-type Salience	
$\phi_{min}^{\mathcal{T}}$	min. salience of entities of type $\mathcal{T}$
$\phi_{max}^{\mathcal{T}}$	max salience of entities of type $\mathcal{T}$
$\phi_{\Sigma}^{\mathcal{T}}$	sum over salience of entities of type $\mathcal{T}$
$\phi_{\mu}^{\mathcal{T}}$	mean salience of entities of type $\mathcal{T}$
Coherence	
φ_{min}	maximum pairwise coherence of all entity pairs
φ_{Σ}	sum over coherence of all entity pairs
φ_{μ}	average coherence of all entity pairs
φ_{μ}^{QTA}	average coherence of entity pairs that involve answer
Answer Type	
$I_{\mathcal{T}}$	binary indicator: answer entity is of type $\mathcal{T}$

Coherence of entity pairs (φ) captures the relative tendency of two entities to appear in the same context. This feature essentially informs us about how much the presence of one entity indicates the presence of the other entity. For example, we would expect a question asking for `BarackObama` using the `WhiteHouse` in the question to be easier than one asking for him using `GrammyAward`. Intuitively, coherence counteracts the effect of salience. Since `BarackObama` is a salient entity, we would expect questions asking for him to be relatively easy. However, asking for him using `GrammyAward` is likely to make the question difficult, as people are unlikely to make a connection between the two entities.

We capture coherence using Wikipedia's link structure. Given two entities e_1 and e_2, we define their coherence as the Jaccard coefficient of the sets of Wikipedia entries that link to their respective entries in Wikipedia. The intuition here is that any overlap corresponds to a mention of the relation between these two entities. For the above measures, we take their maximum, minimum, average, and sum over the question as features as detailed in Table 1.

5 QUERY VERBALIZATION

We now turn to the problem of query verbalization, whereby we transform a query constructed in Section 3 into a natural language question. A human can digest this question without the technical expertise required to understand a query. Our questions test factual knowledge as opposed to linguistic ability. The way a question is formulated is not a factor in predicting its difficulty. This guides our approach to query verbalization, which ensures uniformity in how questions are phrased. Our final goal is to construct well-formed questions that are easy to understand.

We rely on a hand crafted *verbalization template* and automatically generated *lexicons* for transforming a query into a question.

Input: Query, $Q = \{q_1, ..., q_n\}$

$Q_{type} := \{q_i \in Q \mid \text{has the predicate } \texttt{type}\} = \{q_{t_1}, \ldots, q_{t_m}\}$
$Q_{instance} := Q \setminus Q_{type} = \{q_{i_1}, \ldots, q_{i_l}\}$

Which $\texttt{verbalize}(q_{t_1}), \ldots,$ *and* $\texttt{verbalize}(q_{t_m})$
$\texttt{verbalize}(q_{i_1}), \ldots,$ *and* $\texttt{verbalize}(q_{i_l})$?

Figure 3: Verbalization Template

The verbalization template specifies where the different components of the query appear in the question. The lexicon serves as a bridge between knowledge graph entries and natural language.

5.1 Verbalization Template

Our approach to verbalizing queries is based on templates. Such approaches are standard in the natural language generation literature [22, 30]. We adopt a template suitable for a quiz game show given in Figure 3. Most of the work is done in the function `verbalize`.

The function `verbalize` takes a triple pattern and produces its verbalization. How this verbalization is performed depends on the nature of the triple pattern. More concretely, there are three distinct patterns possible in our setting (see Section 3):

- **Type**: if the predicate is `type`, then this results in verbalizing the object, which is a semantic type.
- **PO**: where the triple pattern is of form (`?var p o`) and `p` is not `type`.
- **SP**: where the triple pattern is of form (`s p ?var`) and `p` is not `type`.

By considering these cases individually we ensure that linguistically well-formed verbalizations are created.

5.2 Verbalization Lexicons

Semantic items in the knowledge graph are simply identifiers that are not meant for direct human consumption. It is therefore important that we map each semantic item to phrases that can be used to represent it in a natural language string such as a question.

Entities. To verbalize entities we follow the approach of Hoffart et al. [21] and rely on the fact that our entities come from Wikipedia. We resort to Wikipedia for extracting surface forms of our entities. For each entity e, we collect the surface forms of all links to e's Wikipedia entry. We consider this text to be a possible verbalization of e. The above process extracts many spurious verbalizations of an entity e. To overcome this issue, we associate with each candidate verbalization the number of times it was used to link to e's Wikipedia entry and restrict ourselves to the five most frequent ones, which we add to the lexicon for the entry corresponding to e.

Predicates. Predicate verbalization depends on the pattern in which it is observed (SP or PO). We rely on our large entity-annotated corpus described in Section 2 for mining predicate verbalizations sensitive to the SP and PO patterns. For each triple $(e_1\ p\ e_2) \in KG$, we collect all sentences in our corpus that match the patterns Pat_{SP} = "$e_1\ w_1 ... w_n\ e_2$" (e.g., "`BarackObama` *was born in* `Hawaii`") and Pat_{PO} = "$e_2\ w_1 ... w_n\ e_1$" (e.g., "`Hawaii` *is the birthplace of* `BarackOmaba`") . Following the distant supervision assumption [27], we hypothesize that '$w_1 ... w_n$' is expressing p. The above hypothesis does not always hold. To filter out possible noise we resort to a combination of heuristic filtering and scoring. We remove from the above verbalization candidate set any phrases that are longer than 50 characters or contain a third entity e_3. We subsequently score how good of a fit a phrase '$w_1 ... w_n$' is for a predicate p using normalized pointwise mutual information (npmi). For each predicate p, we retain the 5 highest scoring verbalizations for each of the two patterns, Pat_{SP} and Pat_{PO}, which are used for verbalizing SP and PO triple patterns, respectively.

Types. As explained in Section 2, our types are WordNet synsets. We therefore rely on the lexicon distributed as part of WordNet for type paraphrasing. Each of the three lexicons provides several ways to verbalize a semantic item. We verbalize a semantic item by choosing a verbalization uniformly at random from the corresponding lexicon to ensure variety.

6 MULTIPLE-CHOICE QUESTIONS

The final component in our question generation framework turns a question into a multiple-choice question (MCQ). This has several advantages: in general, it is easier to administer a MCQ as the problem of answer verification can be completely mechanized. In general, where knowledge questions are involved (as opposed to free response questions that might involve opinion), the use of MCQs is widespread as observed in such tests as the GRE.

To turn a question into an MCQ we need *distractors*: entities presented to the user as candidate answers, but are in fact incorrect answers. Of course, not all entities constitute reasonable distractors. Distractors should ideally be related to the correct answer entity and it should generally be possible to confuse a distractor with the correct answer. We call this the *confusability* of a distractor. The more confusable a distractor is with the correct answer, the more likely a test taker is to choose it as an answer, making the MCQ more challenging.

6.1 Distractor Generation

Our starting point for generating distractors is the query $Q = \{q_1, ..., q_n\}$ generated in Section 3, which formed the basis of the question verbalized in Section 5. By starting with a query, we have a fairly simple but powerful scheme for generating distractors. By removing one or more triple patterns from Q we obtain a query $Q' \subset Q$ that has more than one answer entity. All but one of these entities are an incorrect answer to Q.

The relaxation scheme described above can generate a large number of candidate distractors. However, not all relaxations stay close to the original query. If a relaxation deviates too much from Q, the obtained distractors become meaningless. We address this by imposing two restrictions on relaxed queries: (i) a semantic type restriction, and (ii) a relaxation distance restriction.

Semantic type restriction ensures that the answer and distractor are type-compatible. For example, an MCQ asking for a location should not have a person as one of its distractors. The semantic type restriction requires that a semantic type triple pattern is relaxed to the corresponding coarse type.

The relaxation distance restriction refers to relaxations involving instance triple patterns (as opposed to triple patterns specifying type constraints). We define the distance between a query Q and a query $Q' \subseteq Q$ as follows:

$$dist(Q, Q') = |answers(Q')| - |answers(Q)|,$$

where $answers(Q')$ is the set of answers of Q' ($|answers(Q)|$ is always 1). We restrict relaxed queries to have a distance of no more than α, which we set to 10. By pooling the results of all relaxed queries, we form a set of candidate distractors. The choice of distractor is based on how much difficulty we want the distractors to introduce using our notion of distractor confusability.

6.2 Distractor Confusability

An MCQ can be made more or less difficult by the choice of distractors. If one of the distractors is highly confusable with the answer entity, the MCQ is difficult. If none of the distractors is easy to confuse with the answer entity, the MCQ is easy.

Based on this observation we regard a distractor as confusable if it is likely to be the answer to the original question based on our difficulty model. This implies that if an entity is very likely to be the answer to a question asking about a different entity, this entity pair must be similar. We can therefore define confusability between the question's answer e_a and a distractor entity e_{dist} as follows:

$$conf(Q, e_a, e_{dist}) = 1 - |P(diff(Q, e_a) = easy) - P(diff(Q, e_{dist}) = easy)|.$$

Since we can have more than one distractor in an MCQ, we capture the above intuition regarding how multiple distractors affect the overall difficulty of the question. We observe that an MCQ is as confusing as its most confusing distractor and define the confusability of a distractor set $Dist = \{e_{dist1}, e_{dist2}, ...\}$ as:

$$conf(Q, e_a, Dist) = \max_{e_{dist} \in Dist} conf(Q, e_a, e_{dist}).$$

Looking at the big picture, we relate the notion of confusability in an MCQ with our earlier notion of difficulty by combining $diff(Q, e_a) \in \{easy, hard\}$ and $conf(Q, e_a, Dist) \in [0, 1]$. An easy question can be turned into a hard one when a very confusable distractor is added, since the user has to distinguish between two very similar entities. However, adding an easy distractor to a hard question will not change its difficulty because even when both entities are not similar to each other, the user still has to know which entitiy is the correct answer.

7 EXPERIMENTAL EVALUATION

In the following section we evaluate our approach to knowledge question generation from knowledge graphs. We perform two user studies which focus on evaluating the difficulty model and our distractor generation framework.

7.1 Human Assessment of Difficulty

An important motivation for automating difficulty assessment of questions is the fact that it is difficult to judge for the average human what constitutes an easy or hard question. Beinborn et al. [6] has already shown this result for language proficiency tests, where language teachers were shown to be bad at predicting the difficulty of questions when considering the actual performance of students. We would like to observe if the same applies to our setting. To create fair and informative tests, it is crucial that we are able to correctly assess the difficulty of a question.

We start with the assumption that the creators of Jeopardy! are good at automatically assessing question difficulty. Evidence for this was discussed in Section 4, where we showed that there exists a correlation between the monetary value of a question and the likelihood of an incorrect answer by Jeopardy! contestants.

Table 2: Agreement between human evaluators (all measurements are Fleiss' Kappa)

	$eval_2$	$eval_3$	majority
$eval_1$	0.192	0.325	0.500
$eval_2$		0.443	0.661
$eval_3$			0.810

In our experiment we want to show how well the average human can predict the difficulty of a question. To do so, we randomly sampled 100 easy ($200) and 100 hard ($1000) questions from the 500 questions generated in Section 4 to maximize the discrepancy in question difficulty. We then asked three human evaluators ($eval_1$, $eval_2$, $eval_3$) to annotate each of the 200 questions as easy or hard. We then compared their answers with each other and with the ground truth according to Jeopardy!.

Table 2 shows the agreement between each pair of human evaluators and the majority vote difficulty assessment using Fleiss' Kappa [15]. When looking at pairwise agreement between evaluators, it ranges from fair to moderate [24]. This leads us to conclude that it is hard for non-experts to properly judge question difficulty.

We also compared the majority vote of the evaluators on the difficulty of the questions with the ground truth provided by Jeopardy!. The result was agreement on 62.5% of questions. This suggests that there is a need to automate the task.

7.2 Question Difficulty Classification

We start by looking at the quality of our scheme for assigning difficulty levels to questions. The scheme is described in Section 4, where the possible difficulty levels are $D = \{easy, hard\}$. We train our logistic regression classifier on 500 Jeopardy! questions annotated as described in Section 4. Using ten-fold cross validation, our classifier was able to correctly identify the difficulty levels of questions with an accuracy of 66.4%.

To gain insight into how informative our features are, we performed a feature ablation study where we look at the results for all combinations of our features. For this part, we grouped our features into three classes:

- **SAL:** "Salience" features as in Table 1, with additional log-transformation of salience values to deal with long-tail entities.
- **COH:** "Coherence" features in Table 1.
- **TYPE:** "Per-coarse-semantic-type Salience" and "Answer Type" features in Table 1.

Table 3 shows the results of this experiment. Each row corresponds to a certain combination of features enabled or disabled. Rows are shown in descending order of ten-fold cross validation accuracy. It can be seen that best performance is achieved when all of our features are integrated. From this observation it can be reasoned that all features are necessary and give complementary signals. The bottom row corresponds to a random classifier.

Table 3: Ablation study results for features introduced in Section 4. Accuracy is based on ten-fold cross-validation.

SAL	COH	TYPE	Accuracy
yes	yes	yes	**66.4%**
yes	no	yes	65.8%
yes	yes	no	62.6%
yes	no	no	62.2%
no	no	yes	60.0%
no	yes	yes	57.8%
no	yes	no	52.4%
no	no	no	50.0%

7.3 User Study on Difficulty Estimation

In the following we perform an experiment on how well our classifier agrees with relative difficulty assessments of humans for questions generated by our system. It is important to note that we ask humans for relative difficulty assessments as opposed to absolute difficulties, since we have shown in Section 7.1 that humans are not very proficient in judging absolute difficulties.

For the user study we sampled a set of 50 entities with at least 5 non-type facts in Yago. For each entity, we generated a set of three questions and presented them with the answer entity to human annotators. The annotators were asked to order these questions by their relative difficulty and were allowed to skip a set of questions about an entity if they were not familiar with the entity.

We then compared the correlation between the ranking given by each of the human annotators and the output of our logistic regression classifier. For this we used Kendall's τ, which ranges from -1, for perfect disagreement, to 1, for perfect agreement.

A total of 13 evaluators took part in the study and evaluated 92.5 questions on average. Rankings produced by the difficulty classifier moderately agree with the human annotators with $\tau = 0.563$. When the τ-values for users are weighted by study participation, the average rises to $\overline{\tau} = 0.593$. Here, each user's contribution to the final average depends on how many questions she evaluated to avoid overly representing users that evaluated only few questions.

7.4 Distractor Confusability

When generating distractors for MCQs, our goal is to accurately predict the confusability of a distractor given a question's correct answer. In Section 6.2 we presented our scheme for quantifying distractor confusability, which we evaluate here.

For this experiment we automatically generate 10,000 MCQs. Each question has three answer choices, which are the correct answer and two distractors. We then restricted ourselves to 400 MCQs whose distractor pair has the largest difference in confusability. This was done to maximize the probability that study participants can discriminate more confusable from less confusable distractors.

We ran each MCQ through a crowdsourcing platform and asked workers to judge which distractor is more confusing. Each MCQ was judged by 5 workers so we could take the majority vote if judgments where not unanimous. We compare this majority vote with the result of our confusability estimator. Our estimator agreed with the human annotations on 76% of the 400 MCQs. This translates to a Cohen's κ of 0.521, indicating moderate agreement [11].

7.5 Examples

To demonstrate the viability of our approach we provide a few selected examples in Table 4. The table contains generated questions as described in the paper for five topics. For each question Q the verbalization is given, as well as the answer entity e_a and the difficulty of the question being easy $[P(diff(Q, e_a) = easy)]$. Furthermore, we provide two distractors (e_{dist1}, e_{dist2}) with their corresponding confusabilities $[conf(Q, e_a, e_{dist1}), conf(Q, e_a, e_{dist2})]$, where the confusability of $dist1$ is smaller than the confusability of $dist2$. Further examples can be downloaded from http://bit.ly/kg-questions.

The dataset was created by choosing for each topic the ten most salient entities according to our salience measure presented in Section 4. Questions were randomly generated with the constraint that each question should contain at least three triple patterns.

8 RELATED WORK

There has been work on knowledge question generation for testing linguistic knowledge and reading comprehension. The generation of language proficiency tests has been tackled in several works [17, 28, 31]. Here, the focus is on generating cloze (fill-in-the-blank) tests. Beinborn et al. [6] presents an approach for predicting the difficulty of answering such questions with multiple blanks using SVMs trained on four classes of features that look at individual blanks, their candidate answers, their dependence on other blanks, and the overall question difficulty.

Question generation for reading comprehension is aimed at evaluating knowledge from text corpora. This includes including general Wikipedia knowledge [7, 20] and specialized domain such as medical texts [2, 42]. While the above works focus on generating a question from a single document, Questimator [18] generates multiple choice questions from the textual Wikipedia corpus by considering multiple documents related to a single topic to produce a question. Work in this area has mostly taken the approach of overgeneration and ranking [20, 42]. Multiple questions are generated for a given passage using rules. A learned model ranks the questions in terms of "acceptability", where acceptable answers should be sensical, grammatical, and not obvious.

Recent work has started to look at the problem of generating questions, including multiple choice ones, from KGs and ontologies [3, 33, 34, 37]. Strong motivations for studying this problem, compared to question generation from text, are scenarios where structured data is what is available at hand, and the ability to generate deeper, structurally more complex questions. Our system is an end-to-end solution for this problem over a large KG.

In Section 5 we presented a simple approach for query verbalization that sits our needs. The query verbalization problem has been tackled by Ngomo et al. for SPARQL [29], and Koutrika et al. for SQL [23], with a focus on usability. Similar to our approach, these earlier works take a template-based approach to verbalization, which are very widely used on the natural language generation from logical form such as SPARQL queries [22, 30].

Much recent work has focused on keyword search [8] and question answering, rather than generation, from knowledge graphs [5, 12, 26, 36, 40, 43], possibly in combination with textual data [4, 32, 44]. The value of knowledge graphs is that they return crisp answers and allow for complex constraint to answer structurally

Table 4: Examples of generated questions for different topics.

$verbalize(Q)$	$e_a[P(diff(Q, e_a) = easy)]$	$e_{dist1}[conf(Q, e_a, e_{dist1})]$	$e_{dist2}[conf((Q, e_a, e_{dist2})]$
Topic: Theoretical Physicists			
Which scientist is a citizen of Weimar Republic and died in Princeton, New Jersey?	Albert Einstein[0.870]	Aaron Lemonick[0.154]	Grover Cleveland[0.963]
Which physicist was awarded the Nobel Prize in Physics and Goethe Prize?	Max Planck[0.745]	Richard Kuhn[0.683]	Albert Schweitzer[1.000]
Topic: Internet Companies of the United States			
Which company created Android (operating system) and Network Security Services?	Google[0.576]	Open Handset Alliance[0.504]	AOL[0.914]
Which company is located in Washington (state) and was created by the person Jeff Bezos?	Amazon.com[0.243]	Cdigix[0.760]	Starbucks[0.991]

complex questions. Of course, question answering has a long history, with one of the major highlights being IBM's Watson [14], which won the Jeopardy! game show combining both structured and unstructured sources for answering.

One important contribution of our work is an approach to compute the difficulty of questions generated. This topic has received attention lately in community question answering [25, 41], by using a competition-based approach that tries to capture how much skill a question requires for answering. There has also been work on estimating query difficulty in the context of information retrieval [10, 45] to learn an estimator that predicts the expected precision of the query by analyzing the overlap between the results of the full query and the results of its sub-queries.

9 CONCLUSION

We proposed a novel end-to-end approach to the problem of generating quiz-style knowledge questions from knowledge graphs. Our approach addresses the challenges inherent to this problem, most importantly estimating the difficulty of generated questions. To this end, we engineer suitable features and train a model of question difficulty on historical data from the Jeopardy! quiz show, which is shown to outperform humans on this difficult task.

REFERENCES

[1] J! Archive. http://j-archive.com.
[2] M. Agarwal and P. Mannem. Automatic gap-fill question generation from text books. In *BEA*, 2011.
[3] T. Alsubait et al. Generating multiple choice questions from ontologies: Lessons learnt. In *OWLED*, 2014.
[4] H. Bast et al. Semantic Search on Text and Knowledge Bases. *Foundations and Trends in IR*, 10(2-3), 2016.
[5] H. Bast and E. Haussmann. More Accurate Question Answering on Freebase. In *CIKM*, 2015.
[6] L. Beinborn et al. Predicting the Difficulty of Language Proficiency Tests. *TACL*, 2, 2014.
[7] A. S. Bhatia et al. Automatic generation of multiple choice questions using wikipedia. In *PReMI*, 2013.
[8] R. Blanco et al. Effective and efficient entity search in RDF data. In *ISWC*, 2011.
[9] K. D. Bollacker et al. Freebase: a Collaboratively Created Graph Database for Structuring Human Knowledge. In *SIGMOD*, 2008.
[10] D. Carmel and E. Yom-Tov. *Estimating the Query Difficulty for Information Retrieval.* Morgan & Claypool Publishers, 2010.
[11] J. Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37, 1960.
[12] W. Cui et al. KBQA: an Online Template Based Question Answering System over Freebase. In *IJCAI*, 2016.
[13] C. Fellbaum, editor. *WordNet: an Electronic Lexical Database.* MIT Press, 1998.
[14] D. A. Ferrucci. Introduction to "this is watson". *IBM Journal of Research and Development*, 2012.
[15] J. L. Fleiss. Measuring Nominal Scale Agreement among Many Raters. *Psychological Bulletin*, 1971.
[16] E. Gabrilovich et al. FACC1: Freebase annotation of ClueWeb corpora, Version 1, 2013.
[17] D. M. Gates. How to Generate Cloze Questions from Definitions: A Syntactic Approach. In *AAAI*, 2011.
[18] Q. Guo et al. Questimator: Generating Knowledge Assessments for Arbitrary Topics. In *IJCAI*, 2016.
[19] M. A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *COLING*, 1992.
[20] M. Heilman and N. A. Smith. Question Generation via Overgenerating Transformations and Ranking. Technical report, 2009.
[21] J. Hoffart et al. Robust Disambiguation of Named Entities in Text. In *EMNLP*, 2011.
[22] N. Indurkhya et al., editors. *Handbook of Natural Language Processing.* Chapman and Hall/CRC, 2010.
[23] G. Koutrika et al. Explaining Structured Queries in Natural Language. In *ICDE*, 2010.
[24] J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics, Vol. 33*, 1977.
[25] J. Liu et al. Question difficulty estimation in community question answering services. In *EMNLP*, 2013.
[26] V. López et al. Scaling up question-answering to linked data. In *EKAW*, 2010.
[27] M. Mintz et al. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.
[28] A. Narendra et al. Automatic Cloze-Questions Generation. In *RANLP*, 2013.
[29] A.-C. Ngonga Ngomo et al. Sorry, I Don'T Speak SPARQL: Translating SPARQL Queries into Natural Language. In *WWW*, 2013.
[30] E. Reiter et al. *Building Natural Language Generation Systems.* Cambridge University Press, 2000.
[31] K. Sakaguchi et al. Discriminative Approach to Fill-in-the-Blank Quiz Generation for Language Learners. In *ACL*, 2013.
[32] D. Savenkov and E. Agichtein. When a knowledge base is not enough: Question answering over knowledge bases with external text data. In *SIGIR*, 2016.
[33] I. V. Serban et al. Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus. In *ACL*, 2016.
[34] D. Seyler et al. Generating quiz questions from knowledge graphs. In *WWW*, 2015.
[35] D. Seyler et al. Automated question generation for quality control in human computation tasks. In *WebSci*, 2016.
[36] S. Shekarpour et al. Question answering on interlinked data. In *WWW*, 2013.
[37] L. Song and L. Zhao. Domain-specific question generation from a knowledge base. *arXiv*, 2016.
[38] F. M. Suchanek et al. Yago: A Core of Semantic Knowledge. In *WWW*, 2007.
[39] F. M. Suchanek et al. Yago2s: Modular high-quality information extraction with an application to flight planning. In *BTW*, volume 214, 2013.
[40] C. Unger et al. Template-based question answering over RDF data. In *WWW*, 2012.
[41] Q. Wang et al. A regularized competition model for question difficulty estimation in community question answering services. In *EMNLP*, 2014.
[42] W. Wang et al. Automatic question generation for learning evaluation in medicine. In *ICWL*, 2007.
[43] K. Xu et al. What Is the Longest River in the USA? Semantic Parsing for Aggregation Questions. In *AAAI*, 2015.
[44] P. Yin et al. Answering Questions with Complex Semantic Constraints on Open Knowledge Bases. In *CIKM*, 2015.
[45] E. Yom-Tov et al. Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *SIGIR*, 2005.