Automated Quiz Generator

Conference Paper · January 2018

DOI: 10.1007/978-3-319-68385-0_15

3 authors, including:

Amit Shekhar Bongir
College of Engineering, Pune
1 PUBLICATION **6** CITATIONS
SEE PROFILE

Vahida Attar
College of Engineering, Pune
73 PUBLICATIONS **598** CITATIONS
SEE PROFILE

Automated Quiz Generator

Amit Bongir[1], Vahida Attar[1], and Ramanand Janardhanan[2]

[1] College of Engineering, Pune, Shivajinagar, Pune - 411005, India,
bongiras13.it,vahida.comp@coep.ac.in
[2] Choose To Thinq, Pune, India,
ramanand@choosetothinq.com

Abstract. Automated Quiz Generator (AQG) is an extension of the factual question generation system implemented by Michael Heilman, which is generic and therefore applicable to any given domain of discourse in natural language. The extensions mainly include the ability to make MCQs out of generated questions and ranking questions by interestingness of the sentence in the input text from which the respective question was generated. Besides, it has functionality to extract interesting trivia from Wikipedia articles of important entities in the input text. Being domain independent, this system relies on DBpedia - a database of structured content extracted from Wikipedia, the largest general reference work on the Internet.

1 Introduction

1.1 MCQ Generation

Multiple-choice questions (MCQ) are arguably the most popular means of conducting objective tests. An MCQ comprises of:

- **Stem** – the question asked
- **Key** – the correct answer
- **Distractors** – incorrect alternatives to the key

Our system, the Automated Quiz Generator (AQG), passes a given, domain-independent piece of input text to Heilman's factual question generation system [Hei11] which provides a basic set of question-answer pairs which we regard as stem-key pairs of candidate MCQs. Our contribution is towards distractor generation, in which we make use of the Semantic Web technology - DBpedia [Leh+14]. Since DBpedia uses the Resource Description Framework (RDF) [LS99] data model to represent structured information extracted from Wikipedia, the methods we discuss throughout this report are applicable to any database adhering to the RDF specifications.

1.2 Ranking generated questions

Heilman's question generation (QG) system uses an *Overgenerate-and-Rank Framework* [HS09] for question generation. Due to overgeneration of questions,

ranking them according to their worth becomes necessary. Ranking in Heilman's QG system is done mainly on linguistic features of generated question like grammaticality, length, pronoun replacement, etc. and each question is ranked according to the given score, which we'll refer to as its *linguistic score*. In section 3.4 we'll discuss about ways to give an interestingness score to each sentence in the input text. The final score for ranking a question, which we'll refer to as the question's *value*, will be computed using both the question's linguistic score and the interestingness score of the source sentence from which the concerned question was generated.

1.3 Trivia Extraction

Forbes is a renowned media company focussed on business, investing, technology, entrepreneurship, leadership, and lifestyle. Thus, the fact that the kind of readers that visit its website feel a nice welcome with a *Quote of the day* on every visit, works very well towards a critical economic aspect - user engagement. But quotes aren't suitable to every kind of website. A more generic alternative is *Trivia* - interesting facts that are not well-known.
In this report, we discuss methods for extracting trivia about any DBpedia entity from the entity's Wikipedia article. The novelty of our methods lies in the translation of the concepts introduced by David Tsurel *et al.* [Tsu+16] befitting to an RDF database like DBpedia.

2 Related Work

2.1 MCQ Generation

There has been much work regarding MCQ generation from domain ontologies. Papasalouros *et al.* [PKK08] discuss ways of MCQ generation by selecting distractors using *class-based*, *property-based* and *terminology-based* strategies. OntoQue [AlY11] applies the strategies introduced by Papasalouros *et al.* [PKK08] and presents *fill-in the blank* type of questions as stem. Cubric and Tosic [CT11] optimize the strategies introduced by Papasalouros *et al.* [PKK08] and use them in their Protégé[3] [TC09] ontology editor. They further discuss annotation-based strategies using annotated information from domain ontology to generate stems and distractors. They present semantic similarity of distractors to the key as a combination of text and ontological similarity.
An important difference between these and other ontology-based techniques [Lop+15] and AQG lies in the method of constructing stem-key pairs. AQG accepts input text from the user, which acts as a syllabus pertaining to which stem-key pairs are generated using Heilman's QG system [Hei11]. Whereas the other mentioned techniques generate stem-key pairs from the domain ontology and consider no syllabus to confine the generated MCQs to. These systems would require constructing a new ontological database or editing existing

[3] http://protege.stanford.edu/

database to generate syllabus specific MCQs.
We present distractor generation of only entities that are proper nouns as proof-of-concept, since distractor generation techniques based on WordNet for entities that are common nouns have been previously explored [MH03].

2.2 Interestingness and Trivia extraction

David Tsurel *et al.* [Tsu+16] present the concepts of a category's *similarity* to an article tagged by that category and its *cohesiveness*. These concepts were then used to find a category's *trivia-worthiness* for a given article. Our work makes significant use of these concepts for trivia extraction as well as assigning interestingness score to input text sentences. Abhay Prakash *et al.* [Pra+15] propose their approach for automatically mining trivia from unstructured text, as they call it - "Wikipedia Trivia Miner (WTM)". They make use of *unigram*, *linguistic* and *entity* features for their machine learning based *interestingness ranker*. The *unigram* and *entity* features are learnt with the training dataset as human voted trivia retrieved from the Internet Movie Database (IMDb) and as such the WTM is more accurate in extracting trivia related to movies.

3 System Design

The rest of the paper assumes that the reader is familiar with RDF [LS99] and the W3C standardized RDF query language - SPARQL. We elaborate on each of the stages depicted in Figure 1 in the following sections. We also present important SPARQL queries used at various stages which could be run on DBpedia's Virtuoso SPARQL Query Editor[4]. The editor has predefined namespace prefixes[5] and default data set name set to http://dbpedia.org. Therefore, we avoid specifying `PREFIX`es and `FROM <http://dbpedia.org>` in each query.

3.1 Important RDF properties we need

- `rdf:type` - denotes classes the entity is an instance of
  Eg: `dbo:Scientist` is one `rdf:type` of `dbr:Alan_Turing`
- `dct:subject` - denotes topics related to the entity
  Eg: `dbc:Fellows_of_the_Royal _Society` is a `dct:subject` of `dbr:Alan_Turing`

Henceforth we'll refer to `rdf:type` and `dct:subject` as just **types** and **subjects** of an entity respectively.
`dct:subject` is one RDF property that has both forward and backward links [Ber06] in DBpedia . This allows retrieving all entities under a subject in one go, thus improving run-time performance. Every SPARQL query in the following sections has benefited from this aspect.

[4] https://dbpedia.org/sparql
[5] https://dbpedia.org/sparql?nsdecl

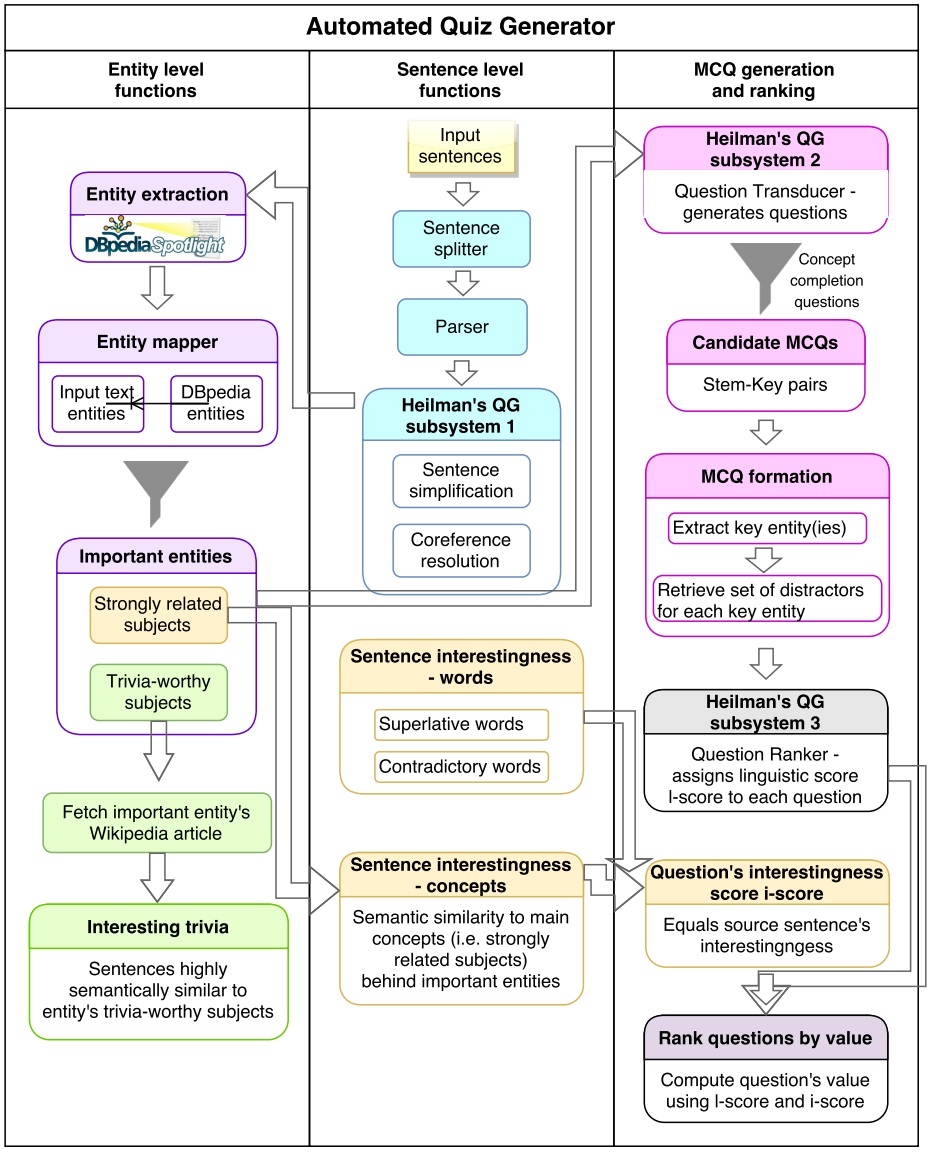

Fig. 1. Architecture of AQG

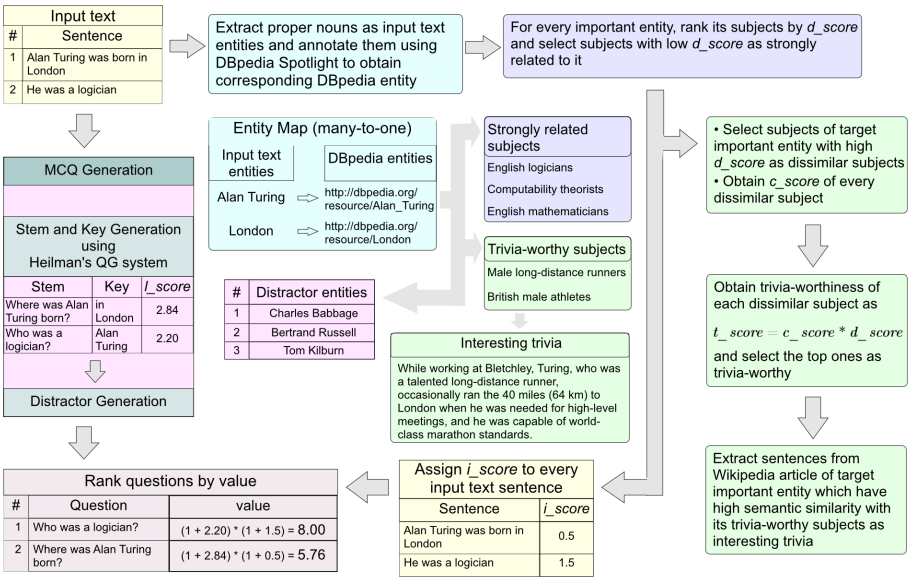

Fig. 2. Overview of AQG with input text related to `Alan Turing` as an example

3.2 Extracting DBpedia entities from input text

The input text is first parsed by the Stanford Parser [K+03]. We then extract named entities in the input text and map them to their corresponding DBpedia entities. For each sentence in input text,

1. According to the TregexPattern[6] [LA06] `NP < NNP & !<< NP`, we extract noun phrases that dominate at least one proper noun and not any other noun phrase, as named entities
2. We apply fuzzy matching techniques to search for a named entity amongst the already mapped entities
3. If the search fails, we annotate the entire sentence using DBpedia Spotlight [Dai+13]. We pass the entire sentence to Spotlight so that it can apply Word Sense Disambiguation (WSD) and annotate even the following named entities in the sentence.

3.3 MCQ Generation

3.3.1 Stem and Key Generation Heilman's question generation (QG) system generates 2 types of *sentence-level* factual questions [Hei11]:

[6] TregexPattern javadoc page: https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/trees/tregex/TregexPattern.html

1. **Concept completion questions** - elicits a given partial concept or proposition
   Eg: *Who served as the first Secretary of State under George Washington?*
2. **Verification questions** - yes-no answer
   Eg: *Was Thomas Jefferson secretary of state?*

The ambitious reader interested in understanding the framework of the QG system is suggested to refer the work of Heilman and Smith [HS09].
We pass the given input text to Heilman's QG system, from which we keep the obtained set of concept completion questions as candidate MCQs, whose question-answer pairs become the stem-key pairs. Heilman's QG system assigns a score to each question based on its linguistic features like grammaticality, vagueness, pronoun resolution, etc., which we'll refer to as its linguistic score *l_score*.

Table 1. Question Generation by Heilman's QG system

#	Question	Answer	Source sentence	*l_score*
1	Where was Alan Turing born?	in London	Alan Turing was born in London	2.84
2	Who was a logician?	Alan Turing	He was a logician	2.20

3.3.2 Distractor Generation The effectiveness of an MCQ is determined by the tendency of its distractors to be selected by the test-taker as an answer. This tendency signifies the distractor quality. An obvious measure of distractor quality is the semantic similarity between the key and concerned distractor. We obtain similarity between two DBpedia entities E_1 and E_2 as

$$sim(E_1, E_2) = |t(E_1) \cap t(E_2)|$$

where $t(E)$ denotes types of entity E. Using the entity map built in section 3.2, we extract DBpedia entities in a candidate MCQ's key, which we'll refer to as *key entities*. For each key entity K, we construct our SPARQL query according to the following procedure:

1. Fetch all DBpedia entities under every subject of K as distractor entities
2. Rank the distractor entities in descending order of their similarity to K

Following is the SPARQL query for the key entity `Alan Turing`

```
SELECT ?distractor, (COUNT(DISTINCT ?type) AS ?similarity)
WHERE {
  dbr:Alan_Turing dct:subject ?subject .
  ?distractor dct:subject ?subject .
```

```
  FILTER (!SAMETERM(?distractor, dbr:Alan_Turing)) .
  dbr:Alan_Turing rdf:type ?type .
  ?distractor rdf:type ?type .
}
GROUP BY ?distractor
ORDER BY DESC(COUNT(DISTINCT ?type))
```

Therefore, for each key entity, we obtain the corresponding set of ranked distractor entities. We make use of a key entity's set of distractor entities as a stack, circularly popping distractor entities every time as the key entity is found in keys of multiple candidate MCQs. The number of distractor entities we pop equals one less than the number of MCQ options desired; one left for the key itself.
We construct a complete distractor by replacing the key entity with the distractor entity. For a candidate MCQ whose key contains multiple key entities, we generate multiple MCQs with common stem-key pair, but with different sets of distractors according to each key entity.

3.4 Ranking generated questions

Summarization is a good strategy in Natural Language Processing (NLP) that provides a summary highlighting important points in an article. The intentions behind our strategy of scoring each sentence by interestingness are similar to those of summarization.

3.4.1 Sentence interestingness and *i_score* We first select important DBpedia entities in the input text. We say an entity to be important if its number of occurrences in the text is at least 50% of that of the most occurring DBpedia entity. One criteria of the interestingness of a sentence is then its semantic similarity to the main concepts behind each important entity. By main concepts, we refer to the strongly related subjects (section 3.4.3) of a DBpedia entity. We use Wordnet-based techniques for computing semantic similarity of a sentence to a subject.
We also infer interestingness of a sentence by the presence of superlative words (like *first*, *best*, etc.) and contradictory words (like *but*, *although*, etc.) [Pra+15].

Table 2. Sentence Interestingness Features

Feature	Weight
Semantic similarity to a strongly related subject of an important entity	0.25
Presence of one or more superlative words	0.50
Presence of one or more contradictory words	0.50

The interestingness score *i_score* of a sentence is then just the sum of the weights of its interestingness features shown in the above table. Note that a weight of 0.25 *each* is added for *every* strongly related subject of an important entity that is semantically similar to the sentence.

3.4.2 Computing question's value The *l_score* of a generated question along with the interestingness score *i_score* (≥ 0) of the sentence in the input text from which the concerned question was generated is used to obtain the question's worth as,

$$value = \begin{cases} (1 + l\_score) * (1 + i\_score), & \text{if } l\_score > 0 \\ l\_score + i\_score, & \text{otherwise} \end{cases}$$

Following are 2 of the ranked MCQs generated for the input text we have considered

Q1) Who was a logician?
1. Charles Babbage
2. Bertrand Russell
3. Alan Turing
4. Tom Kilburn

Answer: 3
Source sentence: He was a logician
Value: 8.00 (*l_score* = 2.20, *i_score* = 1.5)

Q2) Where was Alan Turing born?
1. in York
2. in London
3. in Boston, Lincolnshire
4. in Chichester

Answer: 2
Source sentence: Alan Turing was born in London
Value: 5.76 (*l_score* = 2.84, *i_score* = 0.5)

3.4.3 Strongly Related Subjects The work of Tsurel *et al.* [Tsu+16] is based on extracting trivia-worthy categories of a Wikipedia article. In the context of DBpedia, an article becomes synonymous to a DBpedia entity and categories to the entity's subjects. Therefore, in accordance with the definition of similarity of an article to a category as discussed in [Tsu+16], we obtain the similarity of a DBpedia entity E to one of its subjects S_E as the average similarity between

E and all DBpedia entities (except E of course) under S_E,

$$sim(E, S_E) = \frac{\sum_{\substack{E_i \in S_E \\ E_i \neq E}} sim(E, E_i)}{|S_E| - 1}$$

The relevant SPARQL query for DBpedia entity `Alan Turing` is as follows,

```
SELECT ?subject, (STR(xsd:double(
  xsd:double(COUNT(DISTINCT ?entity))/
  xsd:double(COUNT(?type)))) AS ?d_score)
WHERE {
  dbr:Alan_Turing dct:subject ?subject .
  ?entity dct:subject ?subject .
  FILTER (!SAMETERM(?entity, dbr:Alan_Turing)) .
  dbr:Alan_Turing rdf:type ?type .
  ?entity rdf:type ?type .
}
GROUP BY ?subject
ORDER BY ASC(?d_score)
```

It ranks all subjects of `Alan Turing` according to their dissimilarity score *d_score* (≥ 0), which is nothing but inverse of the similarity score. Setting the maximum *d_score* as 110% of the *d_score* of the least dissimilar subject as cutoff, we select the strongly related subjects.

3.5 Trivia Extraction

We extract interesting trivia about only the important DBpedia entities in the input text (refer section 3.4.1 for importance criteria). This is done by extracting sentences from the Wikipedia article of an important entity which are highly semantically similar to trivia-worthy subjects (section 3.5.1) of the entity. We obtain the URL of the Wikipedia article of a DBpedia entity using its `foaf:primaryTopic` RDF property and therefore the article's text in response to a query sent to Wikipedia's `TextExtracts` API.

3.5.1 Trivia-worthy Subjects We make further use of the set of subjects ranked by dissimilarity of every important entity obtained in section 3.4.3 to select trivia-worthy subjects. Setting the minimum *d_score* as 25% of the *d_score* of the most dissimilar subject as cutoff, we obtain dissimilar subjects $D_{E_{imp}}$s of an important entity E_{imp}, which would be candidate trivia-worthy subjects for E_{imp}.
We then compute the cohesiveness score *c_score* of each $D_{E_{imp}}$. In accordance with [Tsu+16], we obtain cohesiveness of a dissimilar subject as the average similarity between pairs of entities in it. Instead of calculating similarity of entities pairwise, which would have a time complexity of $O(n^2)$, we implement

a linear time algorithm as discussed below.
We first obtain the distinct types occurring across all entities under $D_{E_{imp}}$ excluding E_{imp}'s types,

$$t(D_{E_{imp}}) = (t(E_1) \cup t(E_2) ... \cup t(E_m)) - t(E_{imp}) \mid E_1, ..., E_m \in D_{E_{imp}}$$

We then keep a count of every type in $t(D_{E_{imp}})$ across all entities under $D_{E_{imp}}$. Setting the minimum count for cutoff as 1% of the number of distinct types $|t(D_{E_{imp}})|$, we exclude the types having low count. This signifies that the excluded type had a low contribution towards the average pairwise similarity of entities under $D_{E_{imp}}$. We divide the sum of the counts of the remaining types with $|t(D_{E_{imp}})|$ to get the cohesiveness score of $D_{E_{imp}}$.

$$c\_score(D_{E_{imp}}) = \frac{\sum\limits_{\substack{t_{sim} \in t(D_{E_{imp}}) \\ count(t_{sim}) > 0.01 * |t(D_{E_{imp}})|}} count(t_{sim})}{|t(D_{E_{imp}})| + 1}$$

We add 1 to $|t(D_{E_{imp}})|$ to handle the case $|t(D_{E_{imp}})| = 0$ which happens when $\forall E_i \in D_{E_{imp}},\ t(E_i) \subseteq t(E_{imp})$.
The denominator consists of the number of *distinct types* instead of the number of *entities* under $D_{E_{imp}}$ for averaging since it provides a clearer idea of *incoherence* - more the variation in types across entities, greater is $|t(D_{E_{imp}})|$, and hence lesser the cohesiveness.
The relevant SPARQL query for obtaining cohesiveness of the dissimilar subject `Male long-distance runners` of `Alan Turing` is as follows,

```
SELECT (STR(xsd:double(SUM(?typeMatchCount))/
  xsd:double(?numEntityTypes) + 1) AS ?c_score)
WHERE {
  {
    SELECT (COUNT(DISTINCT ?type) AS ?numEntityTypes)
    WHERE {
      {
        ?entity dct:subject dbc:Male_long-distance_runners .
        ?entity rdf:type ?type .
      }
      MINUS { dbr:Alan_Turing rdf:type ?type . }
    }
  }
  {
    SELECT ?type (COUNT(?type) AS ?typeMatchCount)
    WHERE {
      {
        ?entity dct:subject dbc:Male_long-distance_runners .
        ?entity rdf:type ?type .
      }
      MINUS { dbr:Alan_Turing rdf:type ?type . }
```

```
    }
    GROUP BY ?type
  }
  FILTER (?typeMatchCount > 0.01 * xsd:double(?numEntityTypes)) .
}
GROUP BY ?numEntityTypes
```

Further, according to [Tsu+16], we obtain trivia-worthiness of a subject as,

$$t\_score = c\_score * d\_score$$

The intuition behind this scoring of trivia-worthiness of a subject of an entity is that, while entities under the subject except the target entity are similar to each other, the target entity itself, on an average, is dissimilar to every other entity under the subject. Thus, it follows that the concerned subject is unexpected/surprising for the target entity, and hence trivia-worthy.
We set the minimum t_score for trivia-worthy subjects as 75% of the t_score of the most trivia-worthy subject.

4 Evaluation

4.1 MCQ Generation

We evaluated MCQ generation with input data set as 15 documents related to 15 diverse fields of knowledge, each document consisting of short descriptions about 3 renowned entities in the respective field. Each MCQ is generated with 4 options. In each document, we select only those MCQs for evaluation having a value of at least 40% of that of the most valued question. Further, from MCQs having common stem-key pair, we select the one having the best set of distractors. Thus, a total of 140 MCQs were selected for evaluation.
The worth of distractors in an MCQ varies largely based on the knowledge of the test-taker about the topic on which the MCQ is based. An expert would directly know the answer and hence distractors would be useless. Therefore we take a comparative approach in evaluating the quality of distractors. We categorized each MCQ based on its distractors into one of the following 4 types:

1. **Good** – highly semantically similar to the key
2. **Wrong due to external factors** – external factors mainly include:
   - wrong coreference resolution by Heilman's QG system
   - wrong entity recognition by DBpedia Spotlight

   These errors led to obtaining distractor entities for the wrong entity, and consequently wrong distractors.
3. **Obviously wrong** – these are ones which require no expertise by the test-taker but can be simply eliminated based on metadata provided by the stem. For example - an MCQ which requires a female person as its answer. Presence of words like "she", "her", etc. in its stem provide the hint that the answer is a woman. A male person as a distractor in this case would obviously be wrong.

4. **Another answer** – a distractor that itself is another answer to the question

Table 3. MCQs categorised based on distractors

Category	Percent
Good	57.14
Wrong due to external factors	24.29
Obviously wrong	15.00
Another answer	3.57

The set of distractors of an MCQ is labeled *good* if all distractors in it are so, otherwise it gets labeled by one of the other 3 categories if even one of the distractors belongs to it. The last 2 categories - *obviously wrong* and *another answer* - are due to the flaw that we haven't considered the context of the question while preparing distractors. Reducing them would require methods mainly pertaining to the field of question answering, which were beyond the limitations of this system.
It should be noted that MCQs categorised as *good* don't ensure that they are fit to be used directly in a test. It was observed that **48.75%** of the *good* MCQs required *post-editing*. By post-editing, we mean

- making the MCQ stem/options grammatically correct
- reducing vagueness of the stem
- reducing length of the options by removing unnecessary information repeating in every option. Since distractor generation involved simply replacing key entity with distractor entity, the information surrounding the key entity gets repeated in every option, which could be removed and instead be used to make the stem more informative, hence reducing its vagueness.

Such post-editing doesn't involve modifying distractor entities themselves.
Thus, it is evident from the results that our MCQ generation methods provide assistance rather than replacement to a test-setter.

4.2 Trivia Extraction

We evaluate trivia extracted by AQG for the same entities belonging to 15 diverse fields of knowledge we used for evaluating MCQs. 1-2 facts per entity were generated, making a total of 81 facts for evaluation. The trivia were categorised into one of the following categories and subcategories:

1. **Interesting**
   (a) *Surprising* – a fact that is unexpected to be known of an entity

   (b) *Just didn't know before* – a fact about an entity that may not be so well-known but not surprising for an entity. For example, the fact that "*Mike Tyson was the first heavyweight boxer to simultaneously hold the WBA, WBC, and IBF titles*" may not be well-known but not surprising for a legendary boxer like Mike Tyson. But it is surely surprising that "*To help pay off his debts, Tyson returned to the ring in 2006*".
2. **Not interesting**
   (a) *Lame*
   (b) *Knew before*
3. **Couldn't understand**
   (a) *Vague* – caused by AQG's attempt to keep the trivia as short as possible
   (b) *Was about wrong entity* – caused by the same problems of wrong coreference resolution by Heilman's QG system and wrong entity resolution by DBpedia Spotlight that troubled MCQ generation

Table 4. Results for trivia extraction

Category	Percent	Constituents	
		Type	Percent
Interesting	49.38	Surprising	45.00
		Just didn't know before	55.00
Not interesting	38.27	Lame	45.16
		Knew before	54.84
Couldn't understand	12.35	Vague	60.00
		Was about wrong entity	40.00

In terms of accuracy in obtaining trivia-worthy subjects, AQG has similar performance to Fun Facts [Tsu+16]. The extracted trivia couldn't be compared since Fun Facts doesn't extract the trivia sentence, as we do from the entity's Wikipedia article. The crucial improvements in performance were observed in terms of execution time. While Fun Facts took about 1.45 hours to extract trivia-worthy categories for 2 entities - `Barack Obama` and `Bill Clinton` - AQG obtained the top trivia-worthy category `Grammy Award winners` as obtained by Fun Facts for both entities and extracted the concerned trivia sentences from their respective Wikipedia article in about 3.5 minutes on the same machine.

5 Future Work

The common reasons that affected both MCQ generation and trivia extraction were external factors like wrong coreference resolution by Heilman's QG system and wrong entity recognition by DBpedia Spotlight. Heilman's QG system uses ARKref [OH13] for coreference resolution, and thus AQG will benefit from its improvements. DBpedia Spotlight provides a ranked list of n-best candidate

DBpedia entities as annotations for a named entity in the input text. Currently, we simply select the top most ranked annotation for every input text entity. In doing so, we are being completely dependent on Spotlight's Word Sense Disambiguation (WSD) techniques it applies to the sentence we pass each time we want to get an entity in that sentence annotated. Rather, we could apply WSD techniques on the entire input text natively first, and then select the best candidate amongst the ranked annotations provided by Spotlight. There is also a corelation between coreference resolution and entity recognition by Spotlight that needs to be worked upon:

- Information about recognized entities can be used for coreference resolution
- More the number of coreferences resolved in a sentence, more is the information about the resolved entities available to Spotlight for WSD

Another important improvement would be considering context of an MCQ's stem for distractor selection. In this way, we'll not only be able to reduce the number of distractors that are *obviously wrong* or *another answer*, but also select distractor entities that are not just semantically similar to the key entity but also semantically related to the stem.
Shortening an answer to a question generated by Heilman's QG system to retain prominently the key entity in it and using the removed information to make the question more informative will reduce both vagueness of the question as well post-editing.
Tsurel *et al.* [Tsu+16] have suggested *personalization* of trivia. This is very much needed given the fact that interestingness of a trivia is highly subjective and depends on various parameters specific to a reader.

References

[AlY11] M. Al-Yahya, "Ontoque: A question generation engine for educational assesment based on domain ontologies," in *Advanced Learning Technologies (ICALT), 2011 11th IEEE International Conference on*, IEEE, 2011, pp. 393–395.

[Ber06] T. Berners-Lee. (2006). Backward and Forward links in RDF just as important, [Online]. Available: http://dig.csail.mit.edu/breadcrumbs/node/72 (visited on 04/24/2017).

[CT11] M. Cubric and M. Tosic, "Towards automatic generation of e-assessment using semantic web technologies," *International Journal of e-Assessment*, 2011.

[Dai+13] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes, "Improving efficiency and accuracy in multilingual entity extraction," in *Proceedings of the 9th International Conference on Semantic Systems*, ACM, 2013, pp. 121–124.

[Hei11] M. Heilman, "Automatic factual question generation from text," PhD thesis, Carnegie Mellon University, 2011.

[HS09] M. Heilman and N. A. Smith, "Question generation via overgenerating transformations and ranking," DTIC Document, Tech. Rep., 2009.

[K+03] D. Klein, C. D. Manning, *et al.*, "Fast exact inference with a factored model for natural language parsing," *Advances in neural information processing systems*, pp. 3–10, 2003.

[LA06] R. Levy and G. Andrew, "Tregex and Tsurgeon: Tools for querying and manipulating tree data structures," in *Proceedings of the fifth international conference on Language Resources and Evaluation*, Citeseer, 2006, pp. 2231–2234.

[Leh+14] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, *et al.*, "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia," *Semantic Web Journal*, 2014.

[Lop+15] M. A. Lopetegui, B. A. Lara, P.-Y. Yen, Ü. V. Çatalyürek, and P. R. Payne, "A Novel Multiple Choice Question Generation Strategy: Alternative Uses for Controlled Vocabulary Thesauri in Biomedical-Sciences Education," in *AMIA Annual Symposium Proceedings*, American Medical Informatics Association, vol. 2015, 2015, p. 861.

[LS99] O. Lassila and R. R. Swick, "Resource description framework (RDF) model and syntax specification," 1999.

[MH03] R. Mitkov and L. A. Ha, "Computer-aided generation of multiple-choice tests," in *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing-Volume 2*, Association for Computational Linguistics, 2003, pp. 17–22.

[OH13] B. O'Connor and M. Heilman, "ARKref: A rule-based coreference resolution system," *arXiv preprint arXiv:1310.1975*, 2013.

[PKK08] A. Papasalouros, K. Kanaris, and K. Kotis, "Automatic Generation of Multiple Choice Questions from Domain Ontologies.," in *e-Learning*, Citeseer, 2008, pp. 427–434.

[Pra+15] A. Prakash, M. K. Chinnakotla, D. Patel, and P. Garg, "Did you know?-Mining Interesting Trivia for Entities from Wikipedia.," in *IJCAI*, 2015, pp. 3164–3170.

[TC09] M. Tosic and M. Cubric, "SeMCQ–Protégé Plugin for automatic ontology-driven multiple choice question tests generation," in *Procs of the 11th International Protege Conference*, Stanford Center for Biomedical Informatics Research, 2009.

[Tsu+16] D. Tsurel, D. Pelleg, I. Guy, and D. Shahaf, "Fun Facts: Automatic Trivia Fact Extraction from Wikipedia," *arXiv preprint arXiv:1612.03896*, 2016.