

Clover Quiz: a trivia game powered by DBpedia

Editor(s): Jens Lehmann, University of Bonn, Germany

Solicited review(s): Vinu E. V, University of Luxembourg, Luxembourg; Dhaval Thakker, University of Bradford, United Kingdom; Agis Papantoniou, National Technical University of Athens, Greece

Guillermo Vega-Gorgojo ^{a,b,c,*,**}

^a *Don Naibe AS, Oslo, Norway*

^b *Department of Informatics, University of Oslo, Norway*

^c *School of Telecommunications Engineering, Universidad de Valladolid, Spain*

Abstract DBpedia is a large-scale and multilingual knowledge base generated by extracting structured data from Wikipedia. There have been several attempts to use DBpedia to generate questions for trivia games, but these initiatives have not succeeded to produce large, varied, and entertaining question sets. Moreover, latency is too high for an interactive game if questions are created by submitting live queries to the public DBpedia endpoint. These limitations are addressed in Clover Quiz, a turn-based multiplayer trivia game for Android devices with more than 200K multiple choice questions (in English and Spanish) about different domains generated out of DBpedia. Questions are created off-line through a data extraction pipeline and a versatile template-based mechanism. A back-end server manages the question set and the associated images, while a mobile app has been developed and released in Google Play. The game is available free of charge and has been downloaded by more than 5K users since the game was released in March 2017. Players have answered more than 614K questions and the overall rating of the game is 4.3 out of 5.0. Therefore, Clover Quiz demonstrates the advantages of semantic technologies for collecting data and automating the generation of multiple choice questions in a scalable way.

Keywords: Knowledge extraction, DBpedia, trivia game, multiple choice question, mobile app

1. Introduction

Wikipedia is the most widely used encyclopedia and the result of a truly collaborative content edition process.¹ There are 295 editions of Wikipedia corresponding to different languages, although the English Wikipedia is the largest with more than

5.4 million entries. Articles do not only include free text, but also multimedia content and different types of structured data like so-called infoboxes and category declarations. Wikipedia has an impressive breadth of topical coverage that includes persons, places, organisations, and creative works. The social and cultural impact of Wikipedia is quite significant: it is the fifth most popular website according to Alexa,² while Wikipedia's content is extensively used in education, journalism, and even court cases.³ Despite the vastness and richness of Wikipedia, its content is only accessible

*Corresponding author. E-mail: guiveg@ifi.uio.no

**This work has been partially funded by the Norwegian Research Council through the SIRIUS innovation center (NFR 237898) and BIGMED (IKT 259055) project, as well as by the Spanish State Research Agency (AEI) and the European Regional Development Fund, under project grants SmartLET (TIN2017-85179-C3-2-R) and RESET (TIN2014-53199-C3-2-R).

¹<https://www.wikipedia.org/>

²<http://www.alexa.com/topsites>

³https://en.wikipedia.org/wiki/Wikipedia#Cultural_impact

through browsing and free-text searching. To overcome this limitation, the DBpedia project builds a knowledge base by extracting structured data from the different Wikipedias [15]. As a result, the latest release of the English DBpedia (2016-10) describes 6.6 million entities and contains 1.7 billion triples [21].

DBpedia is the prototypical cross-domain dataset in the Web of Data [10, ch. 3] and is employed for many purposes such as entity disambiguation in natural language processing [14]. An appealing application case is the generation of questions for trivia games from DBpedia. Some preliminary attempts can be found in the literature [5,12,18,19,29], but these initiatives have fallen short due to simple question generation schemes that are not able to produce varied, large, and entertaining questions. Specifically, supported question types are rather limited, reported sizes of the generated question sets are relative low (in the range of thousands), and no user base seems to exist. Moreover, some of these works create the questions by submitting live queries to the public DBpedia endpoint, hence latency is too high for an interactive trivia game, as reported in [18].

The hypothesis of this work is that a semi-automatic approach can produce varied, numerous, and high-quality questions from DBpedia. In a first stage, a human editor can guide the extraction of data by specifying the classes, literals and relations between classes of interest for a particular domain. In a second stage, the generation of questions is driven by a flexible template authoring process, allowing high control in the selection of the entity sets and ample variety in the formulation of question types. Importantly, the produced questions are production-ready, without requiring any kind of post-processing for readability or presentation purposes. The generated questions can then be hosted in a back-end server that meets the latency requirements of an interactive trivia game. The target case is Clover Quiz, a turn-based multiplayer trivia game for Android devices in which two players compete over a clover-shaped board by answering multiple choice questions from different domains. This paper presents the outcomes of this project, including the mobile app and actual usage information of the players that have downloaded the game through Google Play.

The rest of the paper is organized as follows: Section 2 presents the game concept of Clover

Quiz. Section 3 describes the data extraction pipeline, while Section 4 explains the question generation process. The design of the back-end server and the mobile app is addressed in Section 5. Section 6 deals with the actual usage of Clover Quiz, including user feedback and latency measures. Next, Section 7 draws some lessons learned. The paper ends with a discussion and future work lines in Section 8.

2. Game concept

Clover Quiz is conceived as a turn-based multiplayer trivia game for Android devices. In an online match, two players compete over a clover-shaped board. Each player has to obtain the eight wedges in the board by answering questions on different domains. The player with the floor can choose any of the remaining wedges and then respond to a question on the corresponding domain. If the answer is correct, the player gets the wedge and can continue playing, but if it is incorrect, the floor goes to the opponent. Once a player obtains the eight wedges, there is a duel in which each player has to answer the same five questions in a row. The match is over if the player with the eight wedges wins the duel. In other case, this player loses all the wedges and the match continues until there is a duel winner with the eight wedges.

The target audience of Clover Quiz corresponds to casual game players with an Android phone, in the age range of 18-54, high school/university level education, and Spanish- or English-speaking. Importantly, target users are not supposed to know anything about the Semantic Web and do not require a background on Computer Science or Information Technology. Since the game is purposed for mobile devices, user typing should be limited as much as possible. For this reason, Clover Quiz employs multiple choice questions with four options; note that this is also the solution adopted by other mobile trivia games like QuizUp⁴ and Trivia Crack.⁵

Clover Quiz includes questions from the following domains: Animals, Arts, Books, Cinema, Ge-

⁴<https://play.google.com/store/apps/details?id=com.quizup.core>

⁵<https://play.google.com/store/apps/details?id=com.etermax.preguntados.lite>

ography, Music, and Technology – all of them have a good coverage in DBpedia [15] and are arguably of interest to the general public. About the generation of questions, an important design decision is whether to prepare the questions beforehand or to submit live queries to DBpedia. The latter option was discarded due to the complexity of the question generation process and to the stringent requirements of interactive applications (like Clover Quiz) that cannot be met by the public SPARQL endpoint over the DBpedia dataset – queries to the public DBpedia endpoint can easily take several seconds and periods of unavailability are relatively common, according to the tests carried out in the inception phase of the game. Instead, the question set of Clover Quiz is generated in advance and deployed in a back-end server. This architecture corresponds to the crawling pattern employed in some Semantic Web applications [10, ch. 6].

3. Data extraction

The goal of the extraction phase is to gather the data of interest from a knowledge source, e.g. DBpedia, and produce a consolidated dataset that can be easily exploited to generate multiple choice questions. This is accomplished through a series of steps that are graphically depicted in Figure 1. The question set will then be produced programmatically through an off-line process; for this reason the proposed workflow is supported with a collection of scripts coded in Javascript, while all generated input and output files are in JSON format [6]. In the first stage, a human editor prepares a *Domain specification* file with the instructions for retrieving data. Such a file contains a parameters object with the target endpoint (in this case, the URI of the English DBpedia public endpoint⁶), the maximum number of concurrent requests, the languages to gather labels, the value of the *LIMIT* keyword to be used in SPARQL queries, and the time lapse before saving the progress.

The specification file includes the classes to gather entities in a domain of interest, e.g. *Museum*, *Painting*, or *Painter* in Arts. In some cases, there is a one-to-one correspondence between the in-

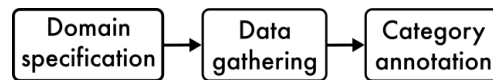


Figure 1. Overview of the data extraction process.

tended concept and a class in the DBpedia ontology, so the editor only has to set the class in the data source, e.g. `dbo:Museum`. Other situations are more involved, requiring the specification of a SPARQL query. This is the case with *Painting*: most of the paintings in DBpedia are of type `dbo:Artwork`, but entities of this type also include sculptures among other things. In addition, there are some paintings like `dbr:Sistine_Chapel_ceiling` which are not of type `dbo:Artwork`. Nevertheless, paintings in DBpedia are annotated with a category narrower than `dbc:Paintings` through the `dct:subject` property. Thus, the SPARQL query in Listing 1 aims to retrieve entities annotated with a subcategory of `dbc:Paintings` and which are of type `dbo:Artwork` or at least have an author/artist or are in a museum.

Listing 1: SPARQL query for retrieving the entities of the *Painting* class

```

select distinct ?X where {
  ?X dct:subject ?S .
  ?S skos:broader{,4} dbc:Paintings .
  { ?X a dbo:Artwork .
    UNION {?X (dbo:author | dbp:author) [] . }
    UNION {?X (dbo:artist | dbp:artist) [] . }
    UNION {?X (dbo:museum | dbp:museum) [] . }
    UNION {[] dbp:works ?X .} . }
  }
  
```

A domain specification file also identifies the literals to be extracted for the entities of a target class – like labels, years, or image URLs – by providing the corresponding datatype properties used in DBpedia. In addition, relations between entities of different classes are also defined; simple cases just involve an object property, e.g. `dct:subject` for getting the Wikipedia categories of *Painting*. Unfortunately, the structure of DBpedia is not very regular and it is common to find alternative properties with similar meaning. As a result, more complicated queries are frequently needed to extract relations between DBpedia entities – see for example the query in Listing 2. This SPARQL query is purposed for gathering the city location of a *Museum* even if different properties are employed to annotate the corresponding entities.

⁶<http://dbpedia.org/sparql>

Listing 2: SPARQL query for retrieving the cities where museums are located

```
select distinct ?entA ?entB where {
  ?entA a dbo:Museum .
  ?entB a dbo:Settlement .
  ?entA (dbo:location | dbp:location |
        dbo:city | dbp:city){1,3} ?entB . }
```

In the *Data gathering* stage, a script takes a specification file as input and queries DBpedia to retrieve the data available of the target domain. Essentially, the script gathers the entities belonging to each class, their literals, and their relations with other entities, as defined in the domain specification file. For every DBpedia entity found, the script also obtains the number of triples with that individual as subject (*outlinks*) and the number of triples with that individual as object (*inlinks*) – these measures are employed to estimate the popularity of an individual in the question generation phase (see Section 4). All queries are paginated using the *LIMIT* and *OFFSET* SPARQL keywords, while a parameter restricts the number of concurrent queries sent to the endpoint. Importantly, the script runs in an incremental way, saving the work in case of errors such as a temporal unavailability of DBpedia. The output of this stage is a file with a JSON object for every entity found, e.g. “The Surrender of Breda” in Listing 3.

The snippet in Listing 3 includes the URI of the entity, the collected literal values, the *outlinks* and *inlinks*, and the relations to entities from other classes. The last item, *painting_categories*, corresponds to the list of Wikipedia categories that the source Wikipedia article is endowed. Wikipedia contributors annotate articles with suitable categories that are organised into a hierarchy that reflects the notion of “being a subcategory of” [4]. Wikipedia categories represent a precious knowledge resource that can be extremely powerful for adding variety and uniqueness to the question set in Clover Quiz. In this running example, the category hierarchy can be exploited to gather the year where “The Surrender of Breda” was completed and other relevant facts such as being a Spanish painting, with animals, and from the Baroque period that are derived from broader categories. To extract these facts, the data extraction pipeline includes a *Category annotation* stage. This involves the authoring of a *Category annotation* file in which a set of Wikipedia categories of interest are specified for the target classes, e.g.

dbc:Baroque_paintings for the *Painting* class. A script takes as input this file and prepares a SPARQL query for gathering the entities of every category of interest. Listing 4 shows the query prepared for obtaining *Baroque paintings*.

Listing 4: SPARQL query for retrieving *Baroque paintings* in the category annotation stage

```
select distinct ?X where {
  ?X dct:subject ?S .
  ?S skos:broader{,4} dbc:Paintings .
  { ?X a dbo:Artwork .
    UNION { ?X (dbo:author | dbp:author) [] . }
    UNION { ?X (dbo:artist | dbp:artist) [] . }
    UNION { ?X (dbo:museum | dbp:museum) [] . }
    UNION { [] dbp:works ?X . } } .
  ?X dct:subject/skos:broader{,4} dbc:Baroque_paintings . }
```

This query is based on the one in Listing 1 (corresponding to the *Painting* class); the only addition is the last property path pattern that includes the category of interest, i.e. *dbc:Baroque_paintings*. Since Wikipedia categories do not behave as a strict taxonomy, a maximum category depth parameter sets a boundary to limit undesired implications. The default value is four, reflecting a trade-off between coverage and strict taxonomy behaviour. Nevertheless, this parameter can be adjusted per case if necessary. In the example above, “The Surrender of Breda” is annotated as *Baroque_paintings* because *dbc:Velazquez_paintings_in_the_Museo_del_Prado* is a subcategory of *dbc:Baroque_paintings*. The obtained annotations are added to the corresponding JSON object, e.g. Listing 5 shows all the annotations made for the running example.

Listing 5: Annotations added to the JSON object in Listing 3 after the data annotation stage

```
"year": 1634,
"Baroque_paintings": true,
"Spanish_paintings": true,
"War_paintings": true,
"Animals_in_art": true
```

Table 1 gives some figures about the number of classes specified, the number of entities extracted from DBpedia, and the size of the annotated data files for each domain in Clover Quiz.

4. Question generation

A multiple choice question consists of a *stem* (the question), a *key* (the correct answer), and

Listing 3: JSON object generated in the data gathering stage for a sample entity

```
{ "uri": "http://dbpedia.org/resource/The_Surrender_of_Breda",
  "label": [ {"es": "La rendición de Breda"}, {"en": "The Surrender of Breda"} ],
  "image": "http://en.wikipedia.org/wiki/Special:FilePath/Velazquez-The_Surrender_of_Breda.jpg",
  "outlinks": "85",
  "inlinks": "2",
  "painter": [ "http://dbpedia.org/resource/Diego_Velázquez" ],
  "museum": [ "http://dbpedia.org/resource/Museo_del_Prado" ],
  "painting_categories": [ "http://dbpedia.org/resource/Category:1634_paintings",
    "http://dbpedia.org/resource/Category:Velazquez_paintings_in_the_Museo_del_Prado",
    "http://dbpedia.org/resource/Category:War_paintings",
    "http://dbpedia.org/resource/Category:Horses_in_art" ] }
```

Table 1

Summary of the data extraction process for the different domains

	Animals	Arts	Books	Cinema	Geo	Music	Tech	TOTAL
# of classes	6	22	9	10	19	17	18	101
# of entities	82,874	223,022	141,621	353,361	251,927	349,443	162,941	1,565,189
Annotated data (MB)	67	108	79	236	122	202	115	929

distractors (a set of incorrect, yet plausible, answers) [2]. In Clover Quiz, the challenge is to produce numerous, varied, and entertaining questions in a scalable way. Moreover, a question difficulty estimator is required to match the questions to the players' skills during the game – intuitively, novice players should get easy questions, while experienced players should get more challenging questions as they progress through the game. To comply with these requirements, a template-based question generator is devised. It consists of a script that takes as input a list of question templates and an annotated data file of a domain, as produced at the end on the data extraction pipeline (see Section 3).

The question generator supports different template types in order to allow the creation of varied questions; Table 2 describes the nine template types with illustrating examples. The first seven template types only involve entities from a class: *Image* relies on the availability of an image in the target entity set; *Boolean*, *Boolean negative*, and *Group* exploit category annotations found in the extracted entities; *Date* uses a date property; *Greatest*, and *Numeric* employ a numeric property. The latter two template types (*Relation* and *Relation negative*) connect entities from two classes.

A question template is just a JSON object with a set of key-values, e.g. Listing 6. Each template

includes its own multilingual stem template (see the *question* field in Listing 6). Replacement of entity labels can be easily added to a stem template such as the example provided in Listing 7. In addition, regular expressions can be defined in a question template to perform fine-grained text transformations; these are especially useful with articles in Spanish since they vary with gender and number (this tends to be easier in English with the article “the”). The core part of a template is the key *class* that defines the entities in the annotated data file to which the template applies; in Listing 6, target entities are members of the *Painting* class and have to include the following JSON keys: *image*, *Baroque_paintings*, and *Animals_in_art*. A template can also specify a *min_score* to filter out those candidates with a lower popularity score – this is computed with this formula: $pop_score = outlinks + 10 * inlinks$.⁷ Inspired by the PageRank algorithm [20], the rationale of the employed popularity score is to differentiate well-known entities from obscure ones (note that the DBpedia PageRank [25] could have

⁷*outlinks* and *inlinks* were obtained in the data extraction pipeline (see Section 3). Note that *outlinks* aggregates literal triples and outgoing RDF links, while *inlinks* only counts incoming RDF links; *inlinks* is multiplied by a factor of 10 in *pop_score* to stress its importance.

Table 2
Supported question templates

Template	Description
Image	Present an image of an entity and ask about its name Stem: <i>Which is the painting of the image?</i> (include image)
Example:	Key: label of a painting Distractors: labels of other paintings in the same set
Boolean	Ask about an entity with a particular boolean property Stem: <i>Which is the modernist building?</i> (optional image)
Example:	Key: label of a modernist building Distractors: labels of other buildings in the same set which are not modernist
Boolean negative	Ask about an entity without a particular boolean property Stem: <i>Which is NOT an Ancient Greek sculptor?</i> (optional image)
Example:	Key: label of a sculptor which is not from Ancient Greece Distractors: labels of Ancient Greek sculptors in the same set
Group	Ask about a particular boolean property of an entity and present a group of options Stem: <i>Which is the artistic style of the painter {{painter.label}}?</i> (optional image)
Example:	Key: label of the artistic style of the mentioned painter, e.g. <i>Baroque</i> Distractors: labels of other artistic styles, e.g. <i>Gothic, Renaissance, Mannerist, Romantic</i>
Date	Ask about a date property of an entity Stem: <i>When was {{painter.label}} born?</i> (optional image)
Example:	Key: mentioned painter's year of birth Distractors: other years (longer intervals if the key date is distant to present time)
Greatest	Ask about the entity with the greatest numeric property of the presented options Stem: <i>Which country has the largest population?</i> (optional image)
Example:	Key: label of the country with the largest population of the presented options Distractors: other countries in the set with fewer population (discard those with populations too close or too far from the key)
Numeric	Ask about a numeric property of an entity Stem: <i>Which is the population of {{city.label}}?</i> (optional image)
Example:	Key: population of the mentioned city Distractors: populations of other cities in the same set (discard those too close or too far from the key)
Relation	Ask about an entity of <i>classA</i> related to an entity of <i>classB</i> Stem: <i>Who is the painter of “{{painting.label}}”?</i> (optional image of the painter or the painting)
Example:	Key: label of the painter which authored the mentioned painting Distractors: labels of other paintings in the same set that did not author the mentioned painting
Relation negative	Ask about an entity of <i>classA</i> not related to an entity of <i>classB</i> Stem: <i>Which castle is NOT in {{country.label}}?</i> (optional image of the castle or the country)
Example:	Key: label of a castle in a different country Distractors: labels of castles in the same set which are in the mentioned country

been used as an alternative to the proposed popularity score). Concerning the rest of the items in the template, *image_prop* identifies the JSON key with the image URL, *topic* is employed for classification purposes, and *dif_level* is a subjective rating of the difficulty of the questions generated with a template – ranging from 0 (very easy) to 10 (very difficult).

Listing 6: Example of an *Image* single class question template

```
{ "question": [ {"en": "Which is the name of this painting with animals?"}, {"es": "¿Cuál es el nombre del cuadro con animales de la imagen?" } ], "class": "Painting.image.Baroque_paintings.Animals_in_art", "min_score": 50, "image_prop": "image", "topic": ["baroque"], "dif_level": 1 }
```

When the template in Listing 6 is evaluated, the question generator first obtains the set of paintings that comply with the requirements, e.g. “The Surrender of Breda”. It will then generate a question for each occurrence by getting the image URL (to support the question) and the label of the painting (this will be the correct answer). Finally, the script will prepare three lists of distractors that correspond to distinct difficulty levels. This is performed by taking a random sample of 50 paintings in the same set, estimating the similarity of each element to the correct answer, discarding the less similar distractors, and finally preparing the three lists. Note that a question is more difficult if the distractors are closer to the correct answer [3], so similarity is computed with a measure based on Jaccard’s coefficient [13] that is defined for every class by providing the array keys, e.g. *painting_categories*, boolean keys, e.g. *Baroque_paintings*, and date-based keys, e.g. *year*, of the target entities. This way, Figure 2(left) shows the question created with the template above when applied to “The Surrender of Breda” painting. Variations of this template can be created very easily, e.g. switching from Baroque to Renaissance paintings, or from animal to still life paintings.

All template types have a similar structure, although double class templates are slightly different. Listing 7 shows the template employed to generate the question in Figure 2(right). This template involves two classes (*Museum* and *Country*)

connected through the property *country*. The stem refers to an entity of type *Museum* (*classA*), while answers are of type *Country* (*classB*) – note that the flow goes from *classA* to *classB*, so the template is not inverse. Since a museum can only be located in one country, the property *country* is annotated as functional in the template – this allows the question generator to silently discard museum entities with several country locations (due to wrong annotations in DBpedia). The rest of elements in the template are similar to the ones in Listing 6. Again, it is very easy to prepare variations of this template, e.g. reversing the question flow and asking about the museum (*classA*) located in a specific country (*classB*) – this new template will thus be inverse. Figure 3 shows additional questions from non-Arts domains.

Listing 7: Example of a *Relation* double class question template

```
{ "question": [ {"en": "Where is the {{classA.label}}?"}, {"es": "¿Dónde está el {{classA.label}}?" } ], "classA": "Museum.image", "classB": "Country.Member_states_of_the_United_Nations", "prop": "country", "inverse": false, "functional": true, "min_scoreA": 300, "min_scoreB": 300, "image_propA": "image", "topic": ["museums"], "dif_level": 0 }
```

After creating the questions associated to a template, the script computes an estimator of the questions’ difficulty. It relies on the popularity of the involved entities (see *pop_score* above) to assign a within-template difficulty score. In this way, a question about a popular entity is considered easier than a question constructed with the same template about a less popular entity. In addition, the proposed question difficulty estimator provides a between-template difficulty correction, e.g. a question about the completion year of a painting is arguably more difficult than asking the name of the same painting, so the *dif_level* (see the description of this metric above) of the latter template should be higher. The template author is thus in charge of assigning an appropriate *dif_level* for a question template. Question difficulty is further tuned through the use of three lists of distractors, as described above. The “hard list” contains distractors closer to the question key, while the “easy list” includes more dis-

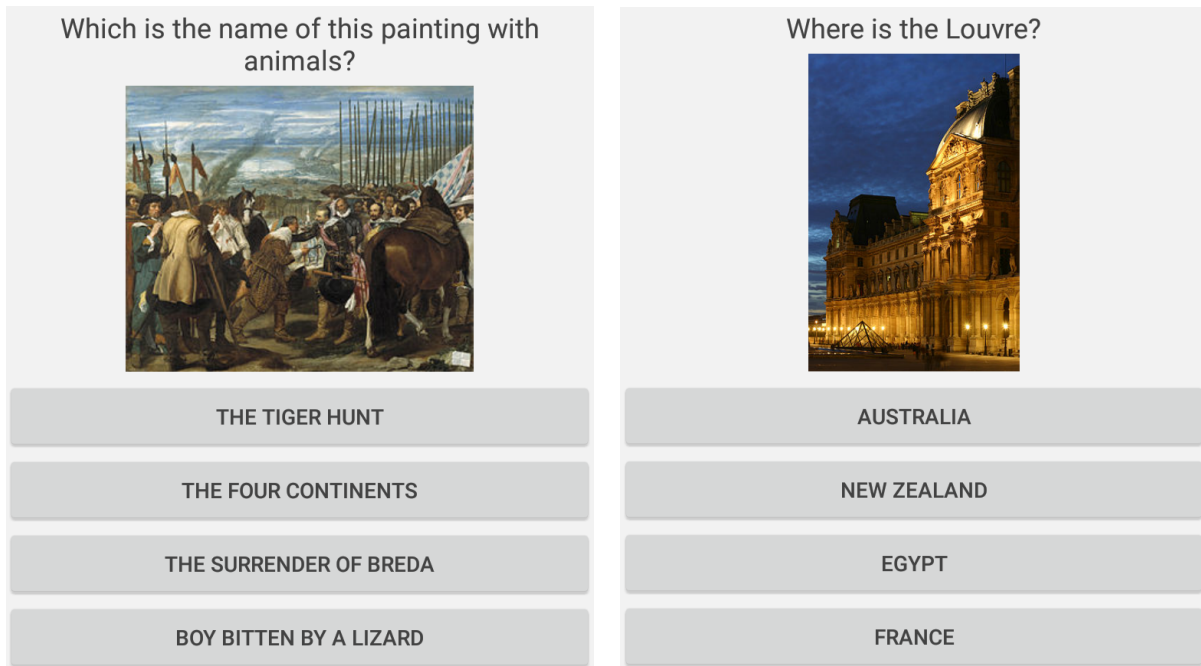


Figure 2. Sample questions from the Arts domain obtained with the mobile app of Clover Quiz. The distractors correspond to the “easy list” – this is especially evident in the second example that includes countries quite dissimilar to France, i.e. non-members of the EU, in different continents, non-French speaking, and so on.

similar distractors. During the game, a player will get distractors from the list that match her expertise in a given topic, thus allowing more balanced games between two participants with disparate expertise levels. With the computed difficulty estimator, questions are then sorted and unique identifiers are given to facilitate their retrieval during the game.


Table 3 presents some aggregated figures of the question set generated for Clover Quiz. The overall process consisted on the creation of several “meta-templates” for every domain (20 to 50, typically) and then preparing the specific templates, e.g. Listings 6 and 7. The rationale is to produce more cohesive questions related to specific topics (like Romanesque, Gothic, Renaissance, Baroque, etc. in Arts) by partitioning the space in smaller and more coherent sets. The downside is that more templates are needed, although the required effort was kept low due to the massive use of copy&paste from the “meta-templates”.

5. Back-end sever and mobile app

After generating the question set of Clover Quiz, the next step is the system design. Figure 4 outlines the overall architecture, split into the mobile app and the back-end server. This separation is purposed to keep the mobile app as lightweight as possible, while the server is in charge of delivering the questions and associated images – note that questions in Clover Quiz are supported with more than 37K low-resolution images, totalling 1.12 GB. To simplify the back-end, a key design decision was to embrace the JSON format to avoid data transformations of the question set, already in JSON. Due to this, a MongoDB database is employed – MongoDB is a scalable and efficient document-based NOSQL system that natively uses JSON for storage.⁸ The question set is thus stored as a collection of documents in MongoDB. Each question is associated with a statistics document containing the number of times the question has been answered correctly, the number of wrong answers, and the number of problems reports.

⁸<https://www.mongodb.com/>

Which is the band from Australia?



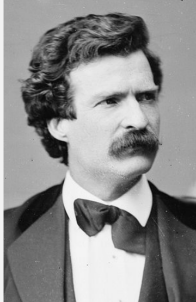
AC/DC

COMMODORES

THE REVOLUTION

THE KINKS

Who wrote "The Adventures of Tom Sawyer"?




EMMA ORCZY

ROBERT LOUIS STEVENSON

PAUL AUSTER

MARK TWAIN

Which is the family of this mammal?




PINNIPED

BEAR

MUSTELID

FELID

What is this?



SMARTPHONE

SMARTWATCH

TABLET COMPUTER

HANDHELD GAME CONSOLE

Figure 3. Sample questions from the Music, Books, Animals and Technology domains obtained with the mobile app of Clover Quiz.

Table 3

Summary of the question generation process for the different domains. There are significantly more English questions in Arts and Books because Spanish labels were missing in many DBpedia entities in these domains

	Animals	Arts	Books	Cinema	Geo	Music	Tech	TOTAL
# of templates	125	269	387	295	724	767	374	2,941
# of questions (Spanish)	15,342	18,121	23,580	49,208	24,086	36,136	21,014	187,487
# of questions (English)	15,347	27,523	46,403	50,199	24,484	36,075	21,017	221,048

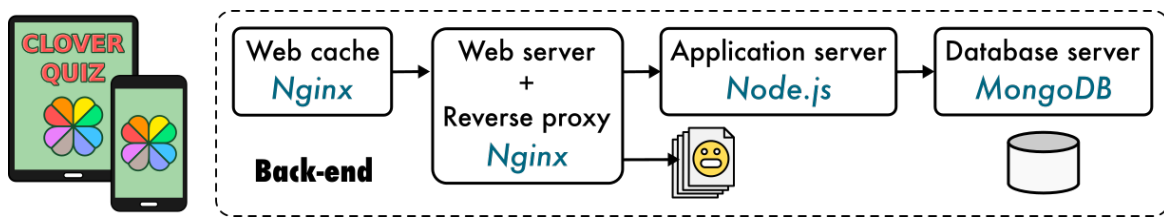


Figure 4. System architecture of Clover Quiz.

The role of the application server is to handle question requests without exposing the database server to the mobile app directly. In this way, the security of the database is not compromised and an eventual upgrade or replacement of the database component does not require changes in the mobile app. Since JSON was derived from JavaScript and is commonly employed with this language, a natural decision was to use Node.js for the application server. Node.js is a popular JavaScript runtime environment for executing server-side code.⁹

Application servers are purposed for handling dynamic content, but they are not very strong for serving static content. Since 67% of the questions in Clover Quiz have an associated image, fast static file serving is an important requirement. This is addressed through the use of Nginx,¹⁰ an efficient and fast performant Web server that excels at serving static content and can also be used as a reverse proxy [22]. Thus, Nginx was configured to host the game images and to forward question requests to the application server. In addition, another Nginx box was set up as a Web cache to improve performance and reduce the back-end load.

Regarding the mobile app, an Android version of the game described in Section 2 was coded. It has been designed following Android conventions

and is structured in three layers: the user interface consists of Android activities and fragments, the domain layer handles user requests and implements the game logic, and the technical services layer addresses data storage, compression, encryption, and network access. The mobile app can be played in phones and tablets and the user interface is built following the Material Design guidelines¹¹ – see sample snapshots in Figure 5. An essential functionality is the matchmaking of players that was implemented using Google Play Games Services¹² that provides a convenient and simple API for turn-based multiplayer games. This way, it is possible to initiate a match against a random player or to invite a friend. The initial screen of the game presents a list of pending invitations, ongoing and finished matches – see Figure 5(left) for an example.

After selecting a match, a clover-shaped board is displayed with eight wedges corresponding to the different domains – see Figure 5(right). The player can push any of the available wedges, e.g. **Music**, then select a subtopic, e.g. **Heavy metal**, and finally answer the question posed. The mobile app keeps a player profile that is used as a basis to select a suitable question; specifically, there are six different expertise levels for each subtopic that controls the difficulty of the questions, along with a randomisation effect. In addition, the player pro-

⁹<https://nodejs.org/>

¹⁰<https://nginx.org/>

¹¹<https://material.io/>

¹²<https://developers.google.com/games/services/>



Figure 5. Sample snapshots of the mobile app of Clover Quiz.

file keeps track of the last 5,000 questions posed to avoid repetitions. Player profiles are saved in the cloud through Google Play Games Services, thus allowing to save players' progression and continue from any device. It is also worth mentioning that the mobile app includes additional features such as a single-player mode, statistics, leaderboards and achievements. Players can also report a problem in a question by simply pushing a "Report problem" button that is always included in the question result screen. This increments the number of problem reports of the associated question statistics document in MongoDB

6. Clover Quiz in practice

The game was released for Android devices on March 11, 2017 under the names 'Clover Quiz' in English and 'Trebial' in Spanish. It is available

free of charge through Google Play¹³ and is part of the catalogue of Don Naípe,¹⁴ a sole proprietorship company specialized in Spanish card games for mobile devices. Clover Quiz was promoted with an in-house ad campaign that ran from March 13 to March 16, i.e. other Android games by Don Naípe¹⁵ showed interstitial ads about Clover Quiz.

At the time of this writing (August 2017), more than 5K users have downloaded the game. Table 4 shows some statistics of the questions answered during this period. It can look striking that most of the requested questions were in Spanish, but this basically reflects the user base of Don Naípe (note that Clover Quiz has been only promoted with in-house ads). Approximately two thirds of the ques-

¹³<https://play.google.com/store/apps/details?id=donnaípe.trebial>

¹⁴<http://donnaípe.com/>

¹⁵<https://play.google.com/store/apps/developer?id=Don+Naípe>

Table 4

Overview of the questions answered from March 11 to August 1

Set	Questions answered	Correct responses	Wrong responses	Problem reports
Spanish	597,771 (100.0%)	382,431 (64.0%)	215,340 (36.0%)	1,915 (0.3%)
English	16,899 (100.0%)	10,106 (59.8%)	6,793 (40.2%)	20 (0.1%)

tions were correctly answered, while each player has taken 122 questions on average, thus indicating a reasonable engagement with the game. Interestingly, the number of problem reports is quite low and concentrated in a small set of questions. A subsequent audit served to spot some problems: a group type template with a wrong option, several animals with misleading images, and one intriguing case in which `Body louse` was classified as a `Primate` – the reason is that this parasite is annotated in Wikipedia with the category `Parasites of humans` that is a subcategory of `Humans`.

Clover Quiz users have also given feedback through Google Play. Specifically, the average rating is 4.3 out of 5.0. The mobile app prompts users to write a review after playing the game for some time, and approximately 1% of them have left a comment. Users’ reviews are generally very supportive: there are some suggestions of new domain areas, e.g. sports, and also a complain about a server failure on April 14, 2017 – there was a system reboot, and the question back-end was not automatically restarted, now it is up again.

The latency of the production back-end server was evaluated in May 2017. `curl`¹⁶ was employed to measure the total response time of 1000 random questions. The client machine ran the experiment in Oslo, while the back-end is deployed in Amsterdam. The average response time was 0.107s with a standard deviation of 0.047s. Similarly, 1000 random images hosted in the back-end were requested with `curl`, taking 0.127s on average with a standard deviation of 0.041s. The reported latencies are quite low and perfectly acceptable for an interactive trivia game. Indeed, Clover Quiz users have not yet complained about performance.

7. Lessons learned

DBpedia is an amazingly comprehensive and vast structured dataset that can be easily con-

sumed and exploited for unforeseen applications, e.g. a trivia game such as Clover Quiz. Despite the remarkable potential of DBpedia as a source of structured knowledge, DBpedia is also messy: there are multiple properties with essentially the same meaning, e.g. `dbp:birthPlace` and `dbp:placeOfBirth`; entities are not always members of the right classes, for example, `dbr:Beyoncé` is not a member of `dbo:MusicalArtist`; classes may be broader than expected, e.g. most of the entities in `dbo:Country` correspond to former countries and empires. As a result, **consuming DBpedia data requires a thorough examination of the target domains** – indeed, Section 3 gives several examples of complicated queries in the data extraction process of Clover Quiz because of this messiness of DBpedia. In this regard, Linked Data browsers and exploration tools like RDF Surveyor [26] can be employed to grasp the data structure. In addition, some curation of the extracted data may also be needed.

A substantial part of the riches of DBpedia correspond to Wikipedia categories. **Wikipedia editors have invested a tremendous effort on the annotation of categories that can be exploited with DBpedia, although special care should be taken to avoid pitfalls**. More specifically, Wikipedia categories are a kind of “folksonomy”, so problems can arise if they are handled as a strict class taxonomy – see the issue with `Body louse` in Section 6. In this respect, the category annotation script employed in Clover Quiz can be configured to limit the number of category levels considered in order to restrict undesired consequences of the category hierarchy. Furthermore, **DBpedia users should be aware that categorisation of entities is unequal, i.e. an entity may not be included in a category although it should**. This is quite challenging for the generation of questions in Clover Quiz, e.g. a question about Baroque paintings would be incorrect if a Baroque painting was wrongly included as a distractor due to an incomplete categorisation. To circumvent this problem, the target sets in Clover Quiz templates are carefully defined to limit the

¹⁶<https://curl.haxx.se/>

impact of missing information; in the previous example, the corresponding template defines a target set comprised of paintings from the Gothic, Renaissance, Baroque and Romantic movements, hence any non-categorised painting is silently discarded.

With respect to the proposed question generation mechanism, several SPARQL queries are needed in order to build a question. For example, the running example in Figure 2(left) requires gathering paintings with animals, painting labels, painting images, inlinks and outlinks. A question of `relation` type like Figure 2(right) requires even more queries, since it involves entities from two classes and their relations. Besides, each query can easily take more than one second, e.g. the query in Listing 1. **In order to meet the stringent latency requirements of an interactive game, the question set is generated off-line.** The proposed data gathering pipeline retrieves the data of interest that is transformed into JSON to simplify the creation of questions programmatically – this solution exemplifies the crawling pattern for consuming Linked Data [10, ch. 6]. As a result, consumer applications can be kept simple and with low latency. The downside is the need of data replication and that applications may work with stale data. Regarding DBpedia, there is already a lag with Wikipedia, since the generation of DBpedia is dump-based with a typical periodicity of 1–2 releases per year [15]. Thus, data freshness can be ensured by re-crawling DBpedia after a new release is available.

Overall, **the template-based mechanism employed to generate the question set is quite flexible, allowing high control in the selection of the target entity sets and providing ample variety in the formulation of questions.** Although the number of templates defined in Clover Quiz is not small (see Table 3), this is mainly due to the creation of multiple template variations. In this way, the class space is partitioned in smaller and more coherent sets, e.g. the template employed to generate question Figure 2(left) is replicated for other movements like Modern Art or Impressionism. About the effectiveness of the devised solution, each English template in the Arts domain served to produce 102 questions in average – and 1,197 per “meta- template” (note that in non-Arts domains the figures are similar). Further, the produced questions are production-ready with

high-quality stems and without requiring any kind of post-processing for readability or presentation purposes.

The proposed question difficulty estimator is based on the popularity of the involved entities in a question to assign a within-template difficulty score. This estimation is further controlled through a between-template difficulty assessment defined by a human editor. In addition, distractor similarity is used to provide three lists of distractors with varying closeness to the question key. **The employed popularity score is a cheap measure for estimating the difficulty of questions that generally works very well**, e.g. The Beatles is the most popular band and United States the most popular country. However, this estimator reproduces similar bias as Wikipedia,¹⁷ for example, the *Ecce Homo* at Borja¹⁸ is an unremarkable painting that became an Internet phenomenon due to a failed restoration attempt – this is the most popular Spanish painting according to the employed popularity score.

8. Discussion

A large body of research has addressed the automatic generation of multiple choice questions, especially using ontologies as a knowledge source – a systematic review on this topic can be found in [1]. The main advantages for using ontologies are their ability to generate deep questions, e.g. questions that asks about relations between the different notions of the domain, and their ability to produce good distractors. However, an ontology is not always available for a domain of interest, thus limiting the applicability of this approach. As an alternative, Linked Data can be used as a source of structured knowledge to generate multiple choice questions. The use of Linked Data for question generation is particularly appealing given the amount of RDF data available. In this regard, DBpedia is one of the preferred knowledge sources due to its quality and breadth of topic coverage.

There are several works in the literature that use DBpedia to generate quiz questions, such

¹⁷https://en.wikipedia.org/wiki/Reliability_of_Wikipedia#Susceptibility_to_bias

¹⁸[https://en.wikipedia.org/wiki/Ecce_Homo_\(Martínez_and_Giménez,_Borja\)](https://en.wikipedia.org/wiki/Ecce_Homo_(Martínez_and_Giménez,_Borja))

as [5,12,16,18,19,23,24,29]. Most of them are early demonstrators that are no longer available. Perhaps the main problem of these initiatives is the use of a simplistic question generation process, e.g. [29] and [12] only support one query type. In addition, none of them exploits Wikipedia categories and supporting images are rarely employed. A notable exception is [5] that invests more effort in the creation of question types by defining subsets of DBpedia and then generating questions (even with images). This approach for question generation is closer to the one devised in Clover Quiz, but it does not scale so well: each question type requires the extraction of a DBpedia subset, as well as changes in the quiz generation engine. In contrast, the approach of Clover Quiz is completely declarative and can be easily ported to other languages. As a result, there are more than 200K questions (in English and Spanish) built from 2.9K templates, while the other initiatives report question sets in the range of thousands.

[1,7,27,28] are examples of question generation systems not purposed for quiz games. [7] uses Linked Data as input, while [1,27,28] employ domain ontologies. [7] only supports two types of multiple choice questions, one based on entity descriptions and another built from verbalising a triple pattern. [1] can produce seven types of questions about ontology classes – this limits its applicability with Linked Data in which entities are the most prominent elements. Finally, [28] is an extended version of the system proposed in [27] that provides an extensive set of question templates that can be potentially used with Linked Data.

Controlling the overall difficulty is a very important feature in question generators, but this aspect is not addressed in many of the surveyed works. Interestingly, [29,24,28] also rely on entity popularity for estimating the difficulty of a question as a basis. [24,28] further control difficulty through an automatic mechanism: [24] analyses the coherence of entity pairs to estimate the difficulty of a question, while [28] assigns a triviality score of the predicates involved in a question stem. The between-template difficulty score used in Clover Quiz plays a similar role, though it relies on the assessment of a human editor. With respect to the generation of distractors in multiple choice questions, DBpedia-based question generators typically use random

distractors, e.g. [29,5,7,12]. Since distractors have an impact on difficulty [9, ch. 41], [3,16,24,27,28] have proposed different approaches to generate suitable distractors – indeed, this is the only mechanism to control question difficulty in [3,16]. In particular, [3] investigates semantics-based distractor generation mechanisms, proposing several measures based on Jaccard’s coefficient [13] to control the difficulty of questions and running a user study to evaluate its effectiveness. Unfortunately, [3] is limited to ontology-based questions that exploit class subsumption, so it cannot be directly applied to generate questions about entities in DBpedia. Nevertheless, Clover Quiz takes inspiration from this work to generate different lists of distractors for distinct difficulty levels, as discussed in Section 4.

On the effort required to produce the question set in Clover Quiz, the most time-consuming tasks correspond to the authoring of the domain specification files and the question templates. The former requires a close inspection of DBpedia to deal with its messiness, as discussed along Section 7. To reduce human effort, a disambiguation tool such as AIDA [11] could be used to find suitable entity types in the knowledge source for a given domain. With respect to the templates, the generation of varied and high-quality questions relies on a thorough template authoring for the selected domains of interest. The approach is indeed scalable, since Clover Quiz is an individual pet project that has been fully carried out during 10 months on a part-time basis. Note that other proposals sometimes rely on the advances of the related domain of question answering over Linked Data [17]. The key challenge of question answering is the translation of information needs into a form suitable for Semantic Web technologies. Thus, [7,24] employ existing natural language generators to verbalise the set of RDF triples that conform a generated question. Natural language conversion is not required in Clover Quiz since every question template includes its own stem template and support for regular expressions, allowing high-control in the produced stems.

Furthermore, an entirely automated question generation pipeline without any configuration step seems unrealistic. Indeed, fully automated approaches normally assume a post-generation step with a human editor in order to improve the verbalisation of the obtained question set. In this re-

gard, the system proposed in [28] underwent an editing phase to do some minor corrections of the generated questions, while [3,7] report grammar problems with their obtained questions in two user studies. Another challenge of automatic generators is the relevancy of the produced questions. Reported user studies with ontology-based approaches show encouraging results in terms of usefulness [3,27,28]. With respect to Linked Data-based approaches, question usefulness is even more challenging given the vastness of data available, but results are somewhat preliminary: [24] reported a crowdsourcing task to filter out irrelevant questions produced with their system; [7] uses the most frequent properties to generate questions from Linked Data, although the effectiveness of this approach is not evaluated in their reported user study; [29] employs property ranking heuristics to generate questions from DBpedia, reporting inconsistencies in DBpedia data and complaints about questions being too simple or too difficult in a user study.

Moving to system design, performance problems are experienced when submitting live queries to DBpedia, as in the case of [18]. Moreover, [7] reports around four seconds to generate a question, while [28] takes several minutes to generate a question set of 25 questions from a domain ontology. To improve latency, some initiatives take small snapshots of DBpedia and run their own triple stores, e.g. [5] and [19]; [12] uses a question cache; and [29] creates the question set off-line. Clover Quiz also adopts the latter approach, but it goes further by transforming the crawled data into JSON to facilitate the generation of questions, in particular to exploit Wikipedia categories. The back-end server in Clover Quiz is able to cope with the requirements of the mobile app, obtaining an average response time of 0.1s in a benchmarking experiment presented in Section 6.

With the release of Clover Quiz as an Android app in Google Play, more than 5K users have downloaded the game and answered more than 614K questions. User ratings are high (4.3 out of 5.0) and comments encouraging, thus suggesting that the questions generated from DBpedia are entertaining and that the game mechanics work. Future work includes the development of an iOS version and a real time mode to further improve user engagement. Moreover, Clover Quiz could be extended to improve DBpedia's content through the

game. Beyond Clover Quiz and DBpedia, the proposed data extraction pipeline and question generator can be used with any other semantic dataset – the only requirement is a SPARQL endpoint. As a result, a promising future line is the generation of multiple choice questions from other knowledge bases; this is especially relevant in the e-learning domain, given the importance of multiple choice questions and the advent of Massive Online Open Courses (MOOCs) [8].

References

- [1] T. Alsubait, Ontology-based multiple-choice question generation, PhD thesis, University of Manchester, United Kingdom, 2015.
- [2] T. Alsubait, B. Parsia and U. Sattler, Generating multiple choice questions from ontologies: lessons learnt, in: *Proceedings of the 11th OWL: Experiences and Directions Workshop (OWLED)*, Riva del Garda, Italy, 2014, pp. 73–84.
- [3] T. Alsubait, B. Parsia and U. Sattler, Ontology-based multiple choice question generation, *Künstliche Intelligenz* **30**(2) (2016), 183–188, DOI: 10.1007/s13218-015-0405-9.
- [4] P. Boldi and C. Monti, Cleansing wikipedia categories using centrality, in: *Proceedings of the 25th International Conference Companion on World Wide Web (WWW '16 Companion)*, Montreal, Canada, 2016, pp. 969–974.
- [5] C. Bratsas, D.E. Chrysou, E. Eftychiadou, D. Kontokostas, P. Bamidis and I. Antoniou, Semantic Web game based learning: An i18n approach with Greek DBpedia, in: *Proceedings of the 2nd International Workshop on Learning and Education with the Web of Data (LiLe 2012)*, Lyon, France, 2012.
- [6] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, Proposed Standard, RFC 7159, The Internet Engineering Task Force (IETF), 2014.
- [7] L. Bühmann, R. Usbeck and A.-C. Ngonga Ngomo, ASSESS – Automatic self-assessment using Linked Data, in: *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*, Bethlehem, PA, USA, 2015, pp. 76–89.
- [8] E. Costello, M. Brown and J. Holland, What questions are MOOCs asking? – An evidence-based investigation, in: *Proceedings of the Fourth European MOOCs Stakeholders Summit (EMOOCs 2016)*, Graz, Austria, 2016, pp. 211–221.
- [9] B.G. Davis, *Tools for teaching*, 2nd edn, John Wiley & Sons, 2009.
- [10] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, Morgan & Claypool, 2011.
- [11] J. Hoffart, M.A. Yosef, I. Bordino, H. Fürstenauf, M. Pinkal, M. Spaniol, B. Taneva, S. Thater and G. Weikum, Robust disambiguation of named entities in text, in: *Proceedings of the 2011 Conference on Em-*

- pirical Methods in Natural Language Processing*, Edinburgh, UK, 2011, pp. 782–792.
- [12] B. Iancu, A trivia like mobile game with autonomous content that uses Wikipedia based ontologies, *Informatica Economica* **19**(1) (2015), 25, DOI:10.12948/issn14531305/19.1.2015.02.
- [13] P. Jaccard, Étude comparative de la distribution florale dans une portion des Alpes et des Jura, *Bulletin de la Société Vaudoise des Sciences Naturelles* **37** (1901), 547–579.
- [14] H. Ji, R. Grishman, H.T. Dang, K. Griffitt and J. Ellis, Overview of the TAC 2010 knowledge base population track, in: *Proceedings of the 3rd Text Analysis Conference (TAC 2010)*, Gaithersburg, MA, USA, 2010.
- [15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer and C. Bizer, DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, *Semantic Web Journal* **6**(2) (2015), 167–195, DOI: 10.3233/SW-140134.
- [16] D. Liu and C. Lin, Sherlock: a Semi-Automatic Quiz Generation System using Linked Data., in: *Proceedings of the 13th International Semantic Web Conference (ISWC 2014)*, Riva del Garda, Italy, 2014.
- [17] V. Lopez, C. Unger, P. Cimiano and E. Motta, Evaluating question answering over linked data, *Web Semantics: Science, Services and Agents on the World Wide Web* **21** (2013), 3–13, DOI: 10.1016/j.websem.2013.05.006.
- [18] F. Mütsch, Auto-generated trivia questions based on DBpedia data, 2017, URL: <https://github.com/n1try/linkedata-trivia>, last accessed June 2018.
- [19] J. Mynarz and V. Zeman, DB-quiz: a DBpedia-backed knowledge game, in: *Proceedings of the 12th International Conference on Semantic Systems (SEMANTICS 2016)*, Leipzig, Germany, 2016, pp. 121–124.
- [20] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, Technical Report, 1999-66, Stanford InfoLab, 1999, Previous number = SIDL-WP-1999-0120.
- [21] S. Praetor, New DBpedia Release – 2016-10, 2017, URL: <http://blog.dbpedia.org/2017/07/04/new-dbpedia-release-2016-10/>, last accessed June 2018.
- [22] W. Reese, Nginx: the high-performance web server and reverse proxy, *Linux Journal* **2008**(173) (2008).
- [23] D. Seyler, M. Yahya and K. Berberich, Generating quiz questions from knowledge graphs, in: *Proceedings of the 24th International Conference on World Wide Web (WWW 2015)*, Florence, Italy, 2015, pp. 113–114.
- [24] D. Seyler, M. Yahya and K. Berberich, Knowledge questions from knowledge graphs, in: *Proceedings of the 3rd ACM International Conference on the Theory of Information Retrieval (ICTIR 2017)*, Amsterdam, Netherlands, 2017.
- [25] A. Thalhammer and A. Rettinger, PageRank on Wikipedia: Towards General Importance Scores for Entities, in: *Proceedings of the 13th European Semantic Web Conference (ESWC 2016)*, Heraklion, Greece, 2016, pp. 227–240.
- [26] G. Vega-Gorgojo, M. Giese and L. Slaughter, Exploring semantic datasets with RDF Surveyor, in: *Proceedings of the 16th International Semantic Web Conference, ISWC 2017*, Vienna, Austria, 2017.
- [27] E.V. Vinu and P.S. Kumar, A novel approach to generate MCQs from domain ontology: Considering DL semantics and open-world assumption, *Web Semantics: Science, Services and Agents on the World Wide Web* **34** (2015), 40–54, DOI: 10.1016/j.websem.2015.05.005.
- [28] E.V. Vinu and P.S. Kumar, Automated generation of assessment tests from domain ontologies, *Semantic Web* **8**(6) (2017), 1023–1047, DOI: 10.3233/SW-170252.
- [29] J. Waitelonis, N. Ludwig, M. Knuth and H. Sack, Whoknows? Evaluating linked data heuristics with a quiz that cleans up DBpedia, *Interactive Technology and Smart Education* **8**(4) (2011), 236–248, DOI: 10.1108/17415651111189478.