Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Drawing Questions from Wikidata

Bachelor Thesis

Swe Geng

gengs@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Philipp Brandes, Laura Peer
Prof. Dr. Roger Wattenhofer

September 4, 2016

Acknowledgements

I thank the Distributed Computing Group, my family and friends for participating in the evaluation and providing feedback. I express my gratitude to Fabian Bissig who introduced me to his work.

Abstract

Quiz applications have been popular on smartphones for entertainment and help improve general knowledge. Questions for these quizzes are created manually. We introduce *Wikidata Quiz*, an automated quiz application that draws its questions from the knowledge base Wikidata. For any chosen topic, we query Wikidata for items related to the topic and create a graph thereof. This graph serves as a basis to generate questions with four answer options and one correct solution. We conduct an evaluation of our algorithm whether it is comparable to a manually generated quiz. Our results show that 50 percent of the questions generated by Wikidata Quiz are rated well.

Contents

Acknowledgements i

Abstract ii

1 Introduction 1

1.1 Related Work . . . 2

1.2 Outline . . . 2

2 Background 3

2.1 Wikidata . . . 3

3 Graph Generation 5

3.1 Fast Graph . . . 5

3.2 Complex Graph . . . 6

3.2.1 Outgoing Claims . . . 8

3.2.2 Qualifiers . . . 8

3.2.3 Incoming Claims . . . 8

3.3 Filtering . . . 8

3.3.1 Geographical Entities . . . 9

3.3.2 Languages . . . 9

3.3.3 Alphabet . . . 9

3.3.4 Generic Terms . . . 10

4 Extraction of Questions 12

4.1 Question Filtering . . . 12

4.1.1 Stop List . . . 13

4.1.2 Similarity . . . 13

4.1.3 Rating System . . . 13

4.1.4 Other Filters . . . 13

4.2 Qualifier Questions 14

5 Extraction and Generation of Answers 15

5.1 Answers in Graph 15

5.2 Additional Answers 16

6 Implementation 17

6.1 Wikibase RDF Query 17

6.2 Python Web Application (Flask) 17

6.3 NetworkX 18

6.4 Redis 18

7 Results 19

7.1 Evaluation of Questions and Feedback 19

7.1.1 Questions extracted from fast or complex Graphs 19

7.1.2 Incomplete or imprecise Data 20

7.1.3 Generic and Umbrella Terms 20

7.1.4 Relevance and off Topic 21

7.2 Rating Comparison 22

7.3 Runtime 22

8 Conclusion and Future Work 24

Bibliography 25

Chapter 1

Introduction

Many mobile Quiz applications have been released on the market in the past and proved to be popular and successful. Millions of users entertain themselves solving countless quiz questions. These questions are commonly designed manually by humans and thus tedious to create. The advantage of manually created questions is that they guarantee a certain quality standard but risk being outdated.

The automated question generation approach has important advantages compared to a manual one. Outdated questions can be regenerated automatically when the knowledge base is updated. Moreover, a wide variety of questions can be generated from the dataset of the knowledge base without manual work involved. Furthermore, multilingual support is possible with a knowledge base, so the questions do not have to be translated manually.

The knowledge base that we chose for our application is Wikidata. Wikidata [1] is an open knowledge base that acts as a central storage for structural data and can be used for free. Wikidata also acts as a collaborative platform to manage data for Wikipedia and its sister projects [2]. It can be read and edited by humans and machines and is constantly updated like Wikipedia. Wikipedia stores information as plain text for human reading. In contrast, Wikidata stores its data in a format, so that machines can easily extract information. Wikidata now allows RDF (Resource Description Framework) exports as has been proposed by Erxleben et al. [3].

Wikidata has grown significantly over the last few years and is also the most edited Wikimedia project [4]. Knowledge bases have the potential to support a variety of fields like automated question generation, question answering (QA) or intelligent personal assistants.

Our quiz application draws questions from the data set of Wikidata related to a topic chosen by the user. We propose an algorithm that generates relevant questions with four possible answer options. We conduct an analysis of user provided feedback to measure the quality of our generated quizzes.

1.1 Related Work

Our work is based on Fabian Bissig's Bachelor thesis [5]. His code serves as a basis on which we implement our improvements and algorithms.

Popular mobile quiz applications include QuizUp [6] and QuizClash [7] that both have over 20 million users. Both games are focused on competing with other players under time constraints over a predefined set of questions. Players also have the possibility to submit their own questions which need to be examined by the developers.

Automatic question generation has been researched in various fields. Brown et al. [8] investigated automatic question generation for vocabulary assessment on a lexical database. Heilman et al. [9] proposed an approach to overgenerate questions and then applying a ranking system using a logistic regression model. Kunichika et al. showed in their work [10] how questions can be automatically generated about an English story. Their work is also part of question answering (QA) which is a discipline in information retrieval and highly related to knowledge bases.

QA is concerned with answering questions in natural language by querying a structured database or knowledge base. Yao and Van Durme proposed question answering in Freebase [11]. Freebase was a knowledge base that was shut down by Google in May 2016 in favor of Wikidata. Yao also proposed a slot-filling method with $< topic, relation, answer >$ tuples to answer a question via a single binary relation [12].

1.2 Outline

In Chapter 2 we explain the key concepts and terms of Wikidata. In order to bring structure to the data of Wikidata we generate a graph which is elaborated in Chapter 3. In Chapter 4 we demonstrate how we extract questions from the generated graph. In Chapter 5 we show how the extraction and generation of answers is conducted. Chapter 6 gives an overview of the technologies used. In Chapter 7 we present the evaluation of the generated questions and a runtime analysis. Chapter 8 gives a summary of our findings and suggests possible future work.

Chapter 2

Background

2.1 Wikidata

Wikidata is a knowledge base that can be accessed under a free license and our choice for our quiz application.

Wikidata stores each *item* with a unique *identifier*. Wikidata items can have *statements* that consist of a *claim* and optional *references*. For instance, the famous television series Game of Thrones is depicted in Figure 2.1 as a Wikidata item with one claim and the identifier Q23572.

Each claim lists a *property* and a *value* which together with the Wikidata item form a *subject-predicate-object* triple where the item is the subject, property the predicate and value the object. The property value can be a linked value which references another Wikidata item (with its unique identifier) or a value of some complex type like integer, string or an URL. Wikidata properties are also labeled with a unique identifier prefixed with a "P". The subject-predicate-object triple for the claim in Figure 2.1 reads Game of Thrones - cast member - Peter Dinklage with P161 identifying cast member and Q310937 referencing Peter Dinklage.

Claims can have an optional *qualifier* which adds additional information to the claim. The term qualifier stands for the combination of the pair qualifier property-qualifier value for additional information as well as only for the qualifier property. In Figure 2.1 the claim includes one qualifier which is character role - Tyrion Lannister and provides additional information. Qualifiers are also properties but can often only be used as qualifiers.

Claims are of utmost importance for extracting information from Wikidata. We distinguish between *outgoing* and *incoming claims* of a Wikidata item. These two terms were originally defined by Fabian Bissig [5]. An outgoing claim is listed under its Wikidata item like cast member - Peter Dinklage is an outgoing claim of Game of Thrones. The Wikidata item is the subject in this case, in contrast to an incoming claim where the item is the object. An example of an incoming claim of Game of Thrones would be Tyrion Lannister - present in work - Game of

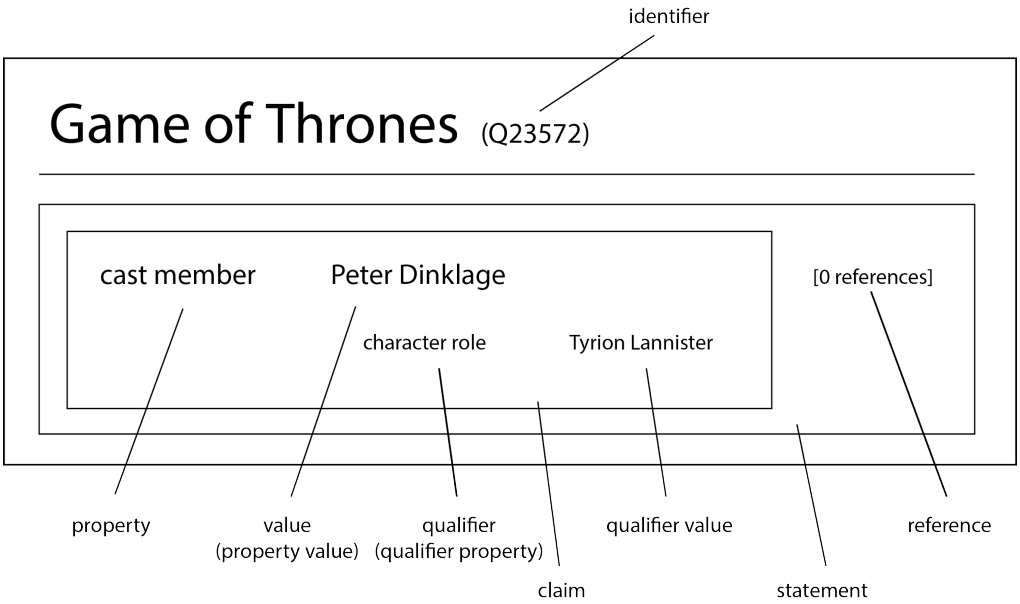

Figure 2.1: Wikidata Schema demonstrated on Game of Thrones

Thrones. To take our example in Figure 2.1, we define the pair Game of Thrones - cast member to be an incoming claim of Peter Dinklage. Incoming claims are not directly listed in Wikidata and need to be queried specifically.

Chapter 3

Graph Generation

In order to generate questions, the data items provided by Wikidata have to be in a structure that shows how the Wikidata items are related. The approach with a directed *graph* has been shown to be useful and efficient by Fabian Bissig [5]. Each vertex in the graph is labeled with an identifier of a Wikidata item, so the vertex and the corresponding Wikidata item can be used interchangeably.

This chapter elaborates how we generate the graph from a Wikidata item selected by the user. We distinguish between *fast graph* and *complex graph* generation. These two graphs are elaborated in Section 3.1 and 3.2, respectively. Section 3.3 describes the filtering process in our graph generation to avoid drifting too far from the original Wikidata item. Drifting too far means that vertices are included to the graph that are deemed irrelevant or not desired by the user. For example, if a user selects Game of Thrones, he does not expect questions about the United States of America.

The fast graph generation ensures that the question generation time is tolerable for the user, as the question generation based on a complex graph can take several minutes. The fast graph will be generated before a complex graph, so that questions can be extracted from a fast graph without waiting for the complex graph. As soon as the complex graph is completed, further questions are extracted and the question set will be updated. The performance of both variants are listed in Section 7.3.

3.1 Fast Graph

We start by defining the Wikidata item chosen by the user as the *origin* vertex in our directed graph.

The outgoing claims of the chosen Wikidata item are then considered, qualifiers and incoming claims are ignored due to their performance impact and long waiting times. All values of the claims that are linked values are added to the graph as vertices and connected to the origin by edges that point away from the

origin. The vertices are labeled according to the values and the edges according to the properties of the respective claims, as shown in Figure 3.1. For instance, the claim Game of Thrones - based on - A Song of Ice and Fire results in two vertices for Game of Thrones and A Song of Ice and Fire connected with a directed edge from Game of Thrones to A Song of Ice and Fire that has the property based on.

Furthermore, each corresponding Wikidata item of a vertex is checked for the existence of an *instance of* property. Most Wikidata items possess one or more instance of properties, such as United States of America is instance of country and sovereign state. If instance of properties exist in a claim of a Wikidata item, then the values of the corresponding claims are added to the graph as vertices and connected with an instance of edge to the respective vertex that represents the Wikidata item. This is depicted with the above example claim United States of America - instance of - country in Figure 3.1.

The vertices which are connected by an instance of edge are essential for generating answers which is discussed in Chapter 5.

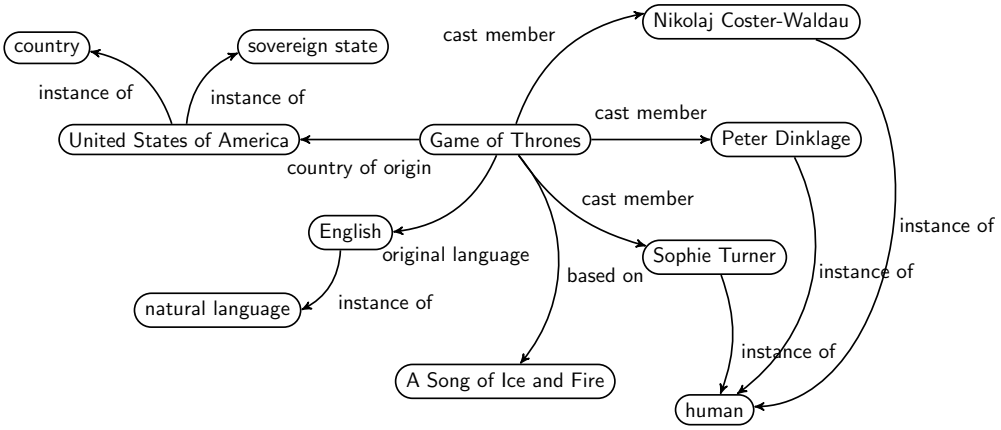

Figure 3.1: Fraction of the Fast Graph of Game of Thrones

3.2 Complex Graph

The generation of the complex graph is based on the algorithm written by Fabian Bissig [5] and is an extension of the fast graph, meaning that the fast graph is a subset of the complex graph.

Again, we start with the selected Wikidata item as origin and consider each claim of the original Wikidata item. Our algorithm uses a breadth-first search which is driven by a queue. As the values of the claims are added to the graph, we also add the corresponding Wikidata items to the queue. After adding all claims of the original Wikidata item to the graph, we proceed to find claims of

the Wikidata items contained in the queue and repeat to add their values to the graph and queue. Our algorithm continues until the queue is empty.

An example is given in Figure 3.2 that illustrates a fraction of the complex graph of Game of Thrones. Note that the fraction is not the same as the fast graph in Figure 3.1, as the fast graph is a subset of the complex graph. Additional nodes and edges are omitted for better visibility. The nodes directly connected to Game of Thrones like Peter Dinklage or Sophie Turner are put into the queue and their claims are further inspected which results in more nodes and edges like actor or Jaime Lannister that are put into the queue again. The edges that have a "Q:" as prefix are qualifiers which are a special case and covered in Subsection 3.2.2.

In order to prevent the algorithm to run indefinitely we define a *level* for each vertex which symbolizes the distance to the origin and a *maximum level*. If the corresponding vertex of our current Wikidata item in the queue has a level that exceeds the maximum level, then the claims of the item are not considered and the item is removed from the queue. Our testing suggests that the maximum level should be defined as 2, as higher levels regularly produce irrelevant questions.

There are several ways of finding appropriate claims to a specific Wikidata item. How our algorithm handles the different types of claims, namely outgoing claims, qualifiers and incoming claims is elaborated in their respective subsections.

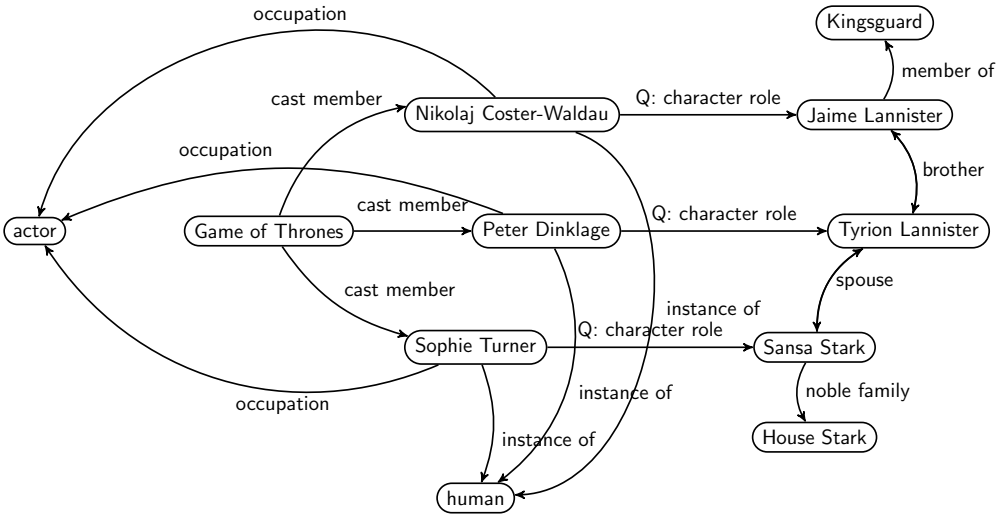

Figure 3.2: Fraction of the Complex Graph of Game of Thrones

3.2.1 Outgoing Claims

This is the most common and most useful type of claim, as they are directly related and thus relevant to the Wikidata item. The Wikidata item represents the subject in an outgoing claim and we receive a set of predicate-object pairs. Our graph generation algorithm creates a new vertex for the object and a new edge connected to the current Wikidata item labeled with the predicate.

3.2.2 Qualifiers

As described in Section 2.1 some Wikidata claims have qualifiers which provide additional information.

For instance, the claim Game of Thrones - cast member - Peter Dinklage has a qualifier character role - Tyrion Lannister. We create a new vertex for Tyrion Lannister and a new edge connected to Peter Dinklage that has the property character role which is depicted in Figure 3.2. This allows our algorithm to put Tyrion Lannister into the queue which can be processed later. However, we do need to mark character role in the graph, as a question Peter Dinklage - character role can be misinterpreted because Peter Dinklage can play other roles than from Game of Thrones. Another example is Khal Drogo - cause of death - murder with qualifier killed by - Daenerys Targaryen. The question murder - killed by makes no sense.

Each edge that is a qualifier is marked and the information of the corresponding property, property value and qualifier value are saved on it.

Question extraction is conducted specifically for qualifiers which is described in Section 4.2.

3.2.3 Incoming Claims

The amount of incoming claims are often humongous, especially for countries like United States of America. Therefore, incoming claims are only considered for the origin vertex if other types of claims yield little results.

3.3 Filtering

Filtering is necessary in our graph generation algorithm to prevent our graph to grow in a direction that is irrelevant to our original Wikidata item and to treat *generic terms* specifically because our regular algorithm often yields few results.

We implement checks for geographical entities, languages and the alphabet that are conducted after a new vertex is created in the graph and before the

corresponding Wikidata item is put into the queue. If a check fails, then the item is not put into the queue but stays in the graph, so that it can still serve as an answer.

These checks are only considered in the complex graph algorithm, since the fast graph algorithm does not propagate further to claims other than those of the origin and has no queue. At the beginning of generating the complex graph, the Wikidata item input by the user is checked for the three characteristics to decide whether an item is relevant at graph generation. The following subsections explain how the respective characteristics are determined and what conditions need to be met in order to fail a check and thus stop the breadth-first search at that item.

3.3.1 Geographical Entities

This check is primarily used to prevent graphs to include a lot of geographical Wikidata items, when the original item is not a geographic entity. For instance, countries are often used in claims, such as Game of Thrones - country of origin - United States of America. This can cause a quiz of Game of Thrones to pose questions about the United States of America and its claims that likely contain other countries, which is not desired.

To determine whether an item is a geographic entity we use the property GeoNames ID which is connected to an identifier in the GeoNames geographical database. If the item is connected with a GeoNames ID property, we assume that it is a geographical entity, otherwise we assume it is not.

If the original item is not a geographical entity and the item considered is one, then the check fails and breadth-first search is stopped for that item.

3.3.2 Languages

Languages often appear in claims like Game of Thrones - original language of work - English and cause the graph to drift in a wrong direction if the original entity is not a language.

We assume that an item is a language if it has a claim with property subclass of that recursively leads to language.

Our check fails if the original item is not a language and the considered item is one.

3.3.3 Alphabet

Sometimes Wikidata items are connected with letters in the alphabet and through further propagation questions like “What follows A?” are generated which are

not desired if the original Wikidata item is not related to the alphabet.

Here we check whether the current item has a claim with a value letter. If this is the case, the item is deemed to be related to the alphabet.

If we conclude that the original item is not related to the alphabet and the considered item is, then our check fails.

3.3.4 Generic Terms

Testing has shown that users often type in a generic term if they are asked for a quiz topic such as film or movie. Unfortunately, those terms usually only have a few claims in Wikidata which results in poor questions or none at all. We assume that if a user types in a generic term, he intends to get a quiz about some famous or relevant items that belong to the generic item.

As stated above the property instance of is defined in Wikidata which exactly refers to an item belonging to some generic term. For example, the movie Titanic has a claim with property instance of and the value film. This allows our algorithm to find all items that belong to a certain generic term.

To determine if the Wikidata item chosen by the user is a generic term, we check if it has incoming claims with the property instance of. Testing has shown that the items with this characteristic are generic terms.

We further assume that a famous or relevant item in Wikidata has more claims than one which is less famous. With this assumption we can generate a list of items that belong to a generic term sorted by their number of claims. We can now take the top items and generate complex graphs thereof and connect them to the original generic term by an instance of edge. However, this ranking procedure results in a big performance impact which is discussed at Section 7.3. An example of the generic graph of film is shown in Figure 3.3 with the complex graphs of the top five items omitted.

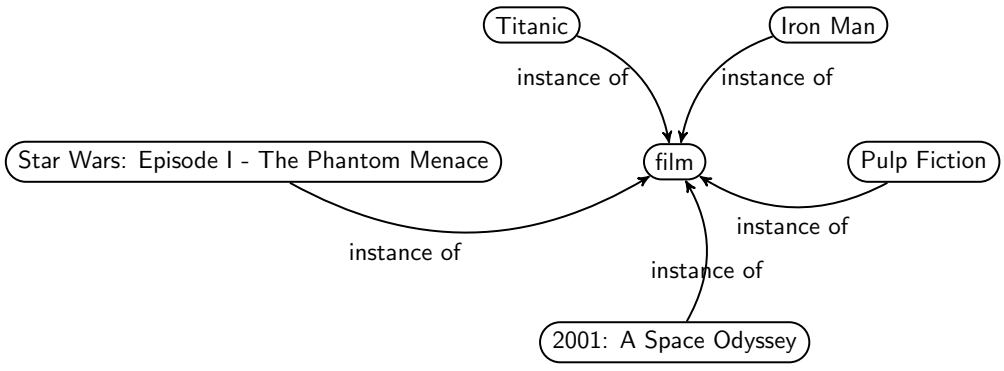

Figure 3.3: Fraction of the Generic Graph of film

To prevent long waiting times for the user, we again implement a fast version by taking some random items that belong to the generic term and apply the fast graph algorithm described in Section 3.1. The fast version for film still looks like the graph in Figure 3.3. However, the films are randomly collected from Wikidata and are probably unknown.

Chapter 4

Extraction of Questions

Our generated graph from Chapter 3 consists of several vertices which represent Wikidata items (e.g. Game of Thrones, English or human) and several edges that represent Wikidata properties (e.g. original language or instance of).

We create *potential questions* in a straightforward manner by taking an edge and its corresponding endpoints, meaning that each edge results in a potential question. Potential questions need to pass all checks to become *final questions* which are displayed to the user. For instance, we take the edge based on connected to Game of Thrones and A Song of Ice and Fire and form the question "Game of Thrones: based on" with the correct answer A Song of Ice and Fire. The question and solution form a triple of subject-predicate-object, such as Game of Thrones - based on - A Song of Ice and Fire where subject and predicate are used for the question and the object represents the *solution* which is the correct answer. We further define *false answer options* as the incorrect answers.

Question 4.1 shows a generated question. The second answer option is the solution and the other three are false answer options.

Q:	Game of Thrones: based on?
1.	The Baroque Cycle
2.	**A Song of Ice and Fire**
3.	The Kent Family Chronicles
4.	Alberta Trilogy

Question 4.1: Example Question about Game of Thrones

4.1 Question Filtering

Testing has shown that some potential questions are irrelevant or trivial for humans to solve and need to be filtered out. In the following subsections we describe our filtering methods to avoid irrelevant questions to be posed. Question filtering is applied to both fast and complex graphs.

4.1.1 Stop List

Some Wikidata properties commonly appear in claims which are not suited for a quiz solved by humans. We have identified three main groups of properties that are not suited for questions:

1. Meta data (e.g. topics main template, described by source, instance of)
2. Too scientific (e.g. parent taxon, taxon rank)
3. Common and incomplete data
   (e.g. diplomatic relation, contains administrative territory)

Properties in these classes are either irrelevant for most users, too difficult to solve or produce false answer options which are actually correct. This can happen due to incomplete data in Wikidata. Instance of properties are extremely common but mostly result in trivial questions and bad false answer options. The properties that fit one of those criteria are listed as stop words and put into a stop list. Properties that are in the stop list are not considered for further question extraction.

4.1.2 Similarity

Sometimes, questions and solutions are too similar and thus trivial for the user. Especially names or counting numbers are often encountered. For example, the question is "Game of Thrones (season 4): part of" and the answer Game of Thrones or "Game of Thrones (season 4): follows" with answer Game of Thrones (season 3). Our question extraction algorithm analyzes the similarity of question and solution by string comparison and deletes a potential question if it is deemed too similar to the correct answer.

4.1.3 Rating System

To detect irrelevant questions and therefore improving the quiz, we implement a rating system where the user has the option to rate a question from 1 to 5.

If sufficient ratings are gathered and the weighted average is below a certain threshold, the potential question will be discarded.

4.1.4 Other Filters

We remove a potential question if a label is missing in the question, solution or false answer options. This is possible, as our application is multilingual and

some Wikidata items or properties do not have labels in the language selected by the user.

Furthermore, we limit the amount of questions by the same subject or predicate. Testing has shown that a lot of questions with the same subject or predicate tend to annoy the user.

4.2 Qualifier Questions

Qualifiers are additional information of a claim and therefore need special consideration when extracting a question because they do not fit the subject-predicate-object scheme. Vertices that are created from qualifiers are marked during the graph generation, so they can be considered during question extraction.

We have to display the subject-predicate-object triple of the claim the qualifier belongs to in order to understand the question. For example, the qualifier reads character role - Jaime Lannister and the subject-predicate-object triple is Game of Thrones - cast member - Nikolay Coster-Waldau. Without the triple, the qualifier does not make sense. In this case, both Game of Thrones and Nikolay Coster-Waldau are important to answer the question character role. We display the question as 4-tuple: "Game of Thrones - cast member - Nikolay Coster-Waldau - character role with the solution Jaime Lannister. The whole question is depicted in Question 4.2.

Q:	Game of Thrones: cast member Nikolay Coster-Waldau: character role?
1.	Steffon Baratheon
2.	Jon Snow
3.	Emmon Frey
4.	**Jaime Lannister**

Question 4.2: Example Question with Qualifier character role

Chapter 5

Extraction and Generation of Answers

The correct answer is given by the question extraction but false answer options need to be found in order to create a quiz. We implement two approaches. The first approach searches false answer options in the graph itself and is applied on complex graphs only. The second approach is primarily used in fast graphs and in case no false answer options can be found in the complex graph.

First and foremost, a false answer option needs to guarantee that it is not correct. Unfortunately, we encountered incomplete data in Wikidata during testing which resulted in false answer options that are actually correct. Therefore, our algorithm can only be as correct as the data in Wikidata itself is.

5.1 Answers in Graph

In order to guarantee an answer option to be false for a particular question, we examine the graph to find a Wikidata item that has one of the following characteristics. The item is not connected to the subject of the question by an incoming edge or possesses such incoming edges but only with different predicates as the question. This indicates that the Wikidata item is a legitimate false answer option according to the data provided by Wikidata. This analysis has been proposed by Fabian Bissig [5].

Moreover, we want to prevent irrelevant false answers that give away the solution and making the question trivial. Fabian Bissig has shown in his work [5] that the predicate instance of is ideal for classifying items and hence finding false answer options with a given solution to the question. Wikidata items can be instances of multiple classes, such as United States of America is instance of country and sovereign state. With this knowledge we search for Wikidata items in our graph that are instances of as many same classes as the solution as possible.

For questions whose solution is an instance of human, we apply two additional checks to further reduce irrelevant answer options. The human class is treated

particularly because Wikidata items that are instance of class human are by far the most common in Wikidata. The first check examines if the predicate of the question is gender related (e.g. mother, sister, brother etc.) and if this is the case, discards all false answer options that do not have the same gender as the solution. The second analysis checks if the false answer option has the property of the corresponding predicate of the question as an incoming edge. For example, the false answer options of question triple Game of Thrones - cast member - Tyrion Lannister need to be instance of human and have an incoming edge with property cast member.

5.2 Additional Answers

For fast graphs the above algorithm is not applicable, as we only examine claims from the origin and as a result get no or few possible false answer option because all vertices are connected with the origin which serves as the only available subject of our questions. The vertices which are connected by an incoming instance of edge are irrelevant because we do not consider instance of as a question as explained in Subsection 4.1.1. Hence, we need another way of gathering false answer options by directly querying Wikidata.

The additional answers algorithm ensures that the false answer options received from Wikidata have at least one instance of in common with the solution but no further checks are conducted. To ensure correctness we still apply the same check as described in Section 5.1. Even though the fast graphs are only a subset of the complex graph, they still connect with all possible Wikidata items on the same level due to breadth-first search. This implies that all correct answers to a specific property are also included in the fast graph, thus the same analysis can be employed to check if an answer option is truly false.

The additional answers algorithm does not guarantee quality but is needed for the answer generation in fast graphs, so that the user receives some questions in a tolerable time span. Moreover, the additional answers algorithm is also applied to questions that do not find any suitable false answer options in the complex graph.

Chapter 6

Implementation

Our application is written in Python and runs on the Flask web framework. The data of Wikidata is stored in a graph database. We use a Python software package for the generation and working with graphs. We implement caching with an in-memory data structure store. The technologies used are the same as in Fabian Bissig's work [5].

6.1 Wikibase RDF Query

Wikibase RDF Query is a software package that provides tools to import Wikidata dumps into Blazegraph, a graph database. The Wikidata dumps are provided in RDF (Resource Description Framework) by Wikidata and are essentially a list of subject-predicate-object triples. Blazegraph is a scalable high-performance graph database which can be queried by SPARQL.

The Blazegraph database and our web application run locally on our server which is equipped with an SSD to improve performance. The Wikidata dumps are compressed to 8GB which inflate into approximately 87GB of indexes.

6.2 Python Web Application (Flask)

We decided to use a web application for our quiz in order to provide universal access for mobile devices. We chose Flask as the web framework which is based on Python and Bootstrap for responsive design.

To query Blazegraph we use SPARQLWrapper which is a Python package that forwards SPARQL queries to a graph database and returns nested dictionaries. The requested data can then be easily extracted.

6.3 NetworkX

NetworkX is a Python software package that provides support for data structures like directed graphs and includes many standard graph algorithms. We use NetworkX to generate, modify and query fast and complex graphs that incorporate Wikidata items and are needed for question extraction.

We use the Graphviz software package that uses the markup language DOT to visualize our graph generated by NetworkX for debugging reasons.

6.4 Redis

Redis is a in-memory data structure store which maps key to values and ideal for caching generated labels and questions.

Keys expire after a defined timeout, so that labels and especially questions are regenerated periodically. As Wikidata is frequently updated, we can regenerate questions, so that they are up to date.

Questions of a Wikidata item are stored in a Redis Set and are copied for each user that accesses that specific item. Then the questions are popped from the user-specific set one after another to be presented to the user. We decided to implement a set because there is no ordering and the questions are shuffled at random. This reduces the chance that the user receives multiple similar questions successively. Testing has shown that this is a desired feature.

The Redis set that stores questions is associated with a key that is the identifier of the respective chosen Wikidata item. Question sets extracted from fast and complex graphs are saved in Redis whereas questions from fast graphs are stored first. The generation of the complex graph and hence its question extraction follow after the questions from the fast graph are saved. After the questions from the complex graph are extracted, they are added to the same Redis set that contains the questions from the fast graph. Duplicates are removed before insertion. The update of the question set for the particular Wikidata item is then propagated to all copies that users accessed, so they can see the newly generated questions.

As our application is multilingual, we cache labels in different languages which are needed by available questions. If a label is not yet in the cache, it is fetched from the local Wikidata dump and put into the Redis store.

Chapter 7

Results

To evaluate our quiz we implement a rating system which was briefly mentioned in Subsection 4.1.3. It allows users to rate each question they solve. Each question can be rated from 1 to 5 which symbolizes terrible to excellent. Fabian Bissig used the same rating scheme in his work [5], so our results are comparable to his.

7.1 Evaluation of Questions and Feedback

Our application was run for several weeks to gather ratings and we also received feedback from participants personally. Overall, we received 1155 ratings with over 40 people participating.

Over 50 percent of our ratings were a 4 or a 5, meaning that on average every other question is a good one. Very few questions were rated both terribly and excellently, so the ratings we received appear to be genuine and make evident that our quiz generation algorithm does produce good questions.

Public evaluation has shown that most people find our quiz interesting and entertaining but also uncovered areas in which our quiz does not perform well.

7.1.1 Questions extracted from fast or complex Graphs

Our rating system does not distinguish between fast and complex questions. By analyzing the log of the ratings, there does not seem to be a difference between the ratings of the first few questions of each topic and those that follow later. We conclude that questions extracted from fast graphs may have worse false answer options but the quality drop is insignificant.

However, we encountered terms which were unheard-of from questions extracted from fast graphs of generic terms. Because the instances are randomly selected as described in 3.3.4, such terms are unfortunately common in fast generic graphs.

7.1.2 Incomplete or imprecise Data

We received feedback that some questions are incorrect where false answer options were actually true. Question 7.1 is a good example to show the issue. In fact, all answer options in this question are legitimate answers, as all belong to characters in Harry Potter. Unfortunately, only Severus Snape of the answer options is listed in the claim Harry Potter - characters on Wikidata and therefore the only solution. Furthermore, Severus Snape is instance of wizard in the Harry Potter universe and our algorithm looks for such instances that are not connected to Harry Potter - characters. Hence, we receive Question 7.1 which is incorrect due to incomplete data in Wikidata.

Q:	Harry Potter: characters?
1.	**Serverus Snape**
2.	Minerva McGonagall
3.	Luna Lovegood
4.	Horace Slughorn

Question 7.1: Incorrect Question about Harry Potter (fantasy literature series)

During testing we encountered an even more peculiar question which is listed as Question 7.2. Through debugging we found out that those "independence days" were from different countries but have the same label. Only the descriptions make those "independence days" distinguishable. Like the previous example, all answer options are instance of the same Wikidata item. In this case it is (another general) independence day.

Q:	United States of America: public holiday?
1.	National Independence Day
2.	**Independence Day**
3.	Independence Day
4.	Independence Day

Question 7.2: Ambiguous Question about the United States of America

Other problematic properties include occupation or genre. Questions that are extracted from these properties often receive bad ratings because their false answer options are frequently ambiguous. An example is given in Question 7.3. Michael Jackson was certainly not a nurse but could be considered a performer or a singer-lyricist, however, composer is the only correct solution.

7.1.3 Generic and Umbrella Terms

Public evaluation has also shown that our algorithm still cannot handle generic or umbrella terms fully. The approach with instance of works well with film or

Q:	Michael Jackson: occupation?
1.	performer
2.	**composer**
3.	singer-lyricist
4.	nurse

Question 7.3: Ambiguous Question about Michael Jackson

human but does not with umbrella terms that do not have direct instances like medicine, science or mathematics. Our results show that generic terms which have no incoming claims with property instance of mostly yield no questions. Further investigation needs to be conducted to find other properties that connect those umbrella terms in order to find suitable questions.

7.1.4 Relevance and off Topic

The biggest issue in our algorithm is still that questions are posed of a Wikidata item that is connected to the chosen topic but are deemed to be irrelevant by the user. Regularly, the user cannot see a relation between these questions and his chosen topic and therefore rates the question badly.

Question 7.4 with the topic chocolate looks irrelevant at first glance but Émile-Justin Menier was a chocolatier and owned the Menier Chocolate company. After some acquisitions, the former Menier Chocolate company is now part of Nestlé. With this knowledge, Question 7.4 seems more relevant.

Q:	Émile-Justin Menier: occupation?
1.	non-fiction writer
2.	entrepreneur
3.	**businessperson**
4.	software engineer

Question 7.4: Bizarre Question about chocolate

Still, knowing that a related item is relevant does not mean that all facts about that item are important. Our algorithm cannot distinguish relevant claims from irrelevant ones except with the use of stop words as described in Subsection 4.1.1.

Our results show that of all questions with ratings 1 or 2 approximately 30 percent have a property associated with human and about 10 percent have a property associated with country. These indicate that questions about humans and countries tend to drift away from the original topic and deemed irrelevant by users. For example, questions about France or Germany appear in a quiz about Switzerland because both are neighboring countries of Switzerland.

7.2 Rating Comparison

Figure 7.1 illustrates the ratings we gathered from our quiz application in green. We gathered 1155 ratings in total with over 40 participants. Fabian Bissig's results [5] of his version 0.5 are depicted in red. He received 1191 ratings for his three versions 0.1, 0.4 and 0.5.

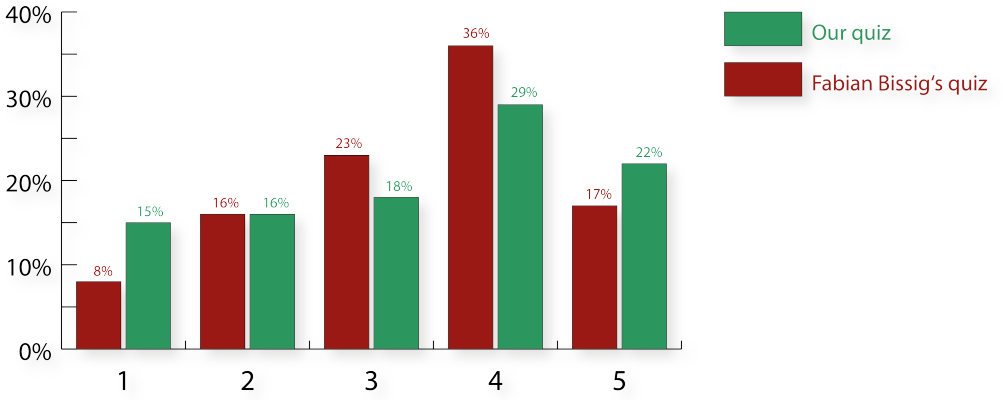

Figure 7.1: Ratings from 1 to 5 (terrible to excellent) compared to Fabian Bissig's Ratings

As can be observed in Figure 7.1, we received similar ratios like Fabian Bissig's ratings. We received many ratings with 1 that is possibly due to high expectations of our participants. Many users were expecting a quiz as good as a manual one because we did not provide several versions and few participants were aware of the previous versions of Fabian Bissig.

We also implemented a simple algorithm to form sentences out of the questions which gave the impression that the quiz was manually created. Unfortunately, participants tended to rate more harshly when the question could not be formulated as a sentence or was badly formulated.

Nevertheless, we could observe that we gathered more ratings with 5 and also received positive feedback that some questions are excellent. According to our results, every fifth question is an excellent one and can compete with manually created questions.

7.3 Runtime

Runtime is an important part for an application querying a knowledge base and a big challenge. Unfortunately, our algorithm is a lot slower compared to the one of Fabian Bissig [5]. This is primarily because of our filtering mechanism described in Section 3.3. We decided to improve quality in favor of performance.

Our algorithm spends the most time on querying Blazegraph over the SPARQLWrapper where the bottleneck lies. Particularly, the generic graph generation causes long waiting times as a GROUP BY is needed in the SPARQL query to rank the Wikidata items by the count of the number of claims.

In Table 7.5 the runtime of the question generation with a fast and complex graph of various popular Wikidata items is depicted. The performance impact to create a generic graph can be clearly seen with the example film. Table 7.5 shows evidently why fast question generation is needed, as we cannot let the user wait a whole minute for a quiz.

Wikidata item	**fast**	**complex**
Game of Thrones	3s	1min 37s
Switzerland	10s	2min 33s
The Simpsons	2s	57s
Albert Einstein	7s	1min 8s
film	9s	9min 50s

Table 7.5: Question generation times of popular Wikidata items

Table 7.6 lists the amount of questions generated out of a fast and a complex graph for the respective Wikidata items. The questions generated from a fast graph are only a fraction of what is extracted from a complex graph.

Wikidata item	**fast**	**complex**
Game of Thrones	28	121
Switzerland	35	89
The Simpsons	24	104
Albert Einstein	67	154
film	47	302

Table 7.6: Number of questions generated of popular Wikidata items

Chapter 8

Conclusion and Future Work

With Wikidata Quiz we introduced an application and algorithms to draw questions from Wikidata automatically. Our algorithm uses a graph to structure the data received from specific Wikidata items and extract questions and answer options thereof. We built a website according to responsive design to give our users a good mobile experience with our quiz application.

Our ratings suggest that our application can generate questions that can compete with manually created ones. However, there are still many that are deemed irrelevant. Our results indicate that Wikidata still has a lot of incomplete and imprecise data. We showed that questions from generic terms can be created and an approach to generate questions fast.

Wikidata has grown over 50 percent in the past year according to our Wikidata RDF dumps and will grow even more in the future. During the development of our application we experienced that an update in Wikidata's data set does impact our quiz and makes it more precise. Furthermore, our quiz application can be used to detect incomplete Wikidata items in an entertaining manner.

We propose that future work invests in creating more diverse questions with values other than linked values like strings or integers. Questions could be posed with aggregates over these values like highest or greatest. Moreover, generic and umbrella terms are still poorly included and would be a valuable addition.

Bibliography

[1] Wikimedia Foundation: Wikidata. https://www.wikidata.org/ Accessed: 2016-08-30.

[2] Vrandečić, D.: Wikidata: A new platform for collaborative data collection. In: Proceedings of the 21st International Conference on World Wide Web, ACM (2012)

[3] Erxleben, F., Günther, M., Krötzsch, M., Mendez, J., Vrandečić, D.: Introducing wikidata to the linked data web. In: International Semantic Web Conference, Springer (2014) 50–65

[4] Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM (10) (2014) 78–85

[5] Bissig, F.: Drawing questions from wikidata (October 2015)

[6] Plain Vanilla Games: QuizUp. https://www.quizup.com/ Accessed: 2016-08-30.

[7] FEO Media AB: QuizClash. http://www.quizclash-game.com/ Accessed: 2016-08-30.

[8] Brown, J.C., Frishkoff, G.A., Eskenazi, M.: Automatic question generation for vocabulary assessment. In: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2005) 819–826

[9] Heilman, M., Smith, N.A.: Good question! statistical ranking for question generation. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Association for Computational Linguistics (2010) 609–617

[10] Kunichika, H., Katayama, T., Hirashima, T., Takeuchi, A.: Automated question generation methods for intelligent english learning systems and its evaluation. In: Proc. of ICCE. (2004)

[11] Yao, X., Van Durme, B.: Information extraction over structured data: Question answering with freebase. In: ACL (1), Citeseer (2014) 956–966

[12] Yao, X.: Lean question answering over freebase from scratch. In: Proceedings of North American Chapter of the Association for Computational Linguistics. (2015) 66–70